THERMODYNAMIC WEIGHT DECAY: GLOBAL VS. BLOCK-WISE CONTROL AROUND CRITICALITY

Anonymous authors
Paper under double-blind review

ABSTRACT

Weight decay is typically specified as a fixed coefficient whose selection requires task-specific search. Inspired by dissipative structures in nonequilibrium thermodynamics, we propose Thermodynamic Weight Decay (TWD), a dynamic weight-decay method that views learning as a balance between energy inflow and dissipation. TWD treats the squared weight norm of each block as internal energy and assigns an architecture-aware reference energy to each block. It defines energy quantities at the granularity of functional blocks and drives each block's internal energy toward a target obtained by scaling its reference energy by a constant factor. On modular addition and multiplication, TWD reduces the number of epochs to grokking onset by a factor of approximately 60 relative to no control and performs comparably to a static weight decay tuned by search. Relative to global control, which constrains only total energy, block-wise control suppresses distortions in the allocation of energy across blocks and reduces the number of epochs to grokking onset by a factor of approximately 20. On Fashion-MNIST and CIFAR-10, it reduces test loss and the generalization gap while largely preserving test accuracy. Application to GPT-2 (124M) confirms that functional-block-wise control remains feasible at this scale. A broad sweep of the target-energy scale factor reveals four empirical regimes, including high plasticity on the low-energy side, lazy feature learning on the high-energy side, and a critical band in which plasticity and stability are balanced. These results show that the level and allocation of weight energy strongly affect plasticity and stability during learning and may serve as controllable indicators for achieving fast and stable learning.

1 INTRODUCTION

Weight decay is one of the most fundamental regularizers for improving generalization by constraining weight norms (Krogh & Hertz, 1992; Loshchilov & Hutter, 2019). Yet, although the weights—the state being controlled—continue to change throughout training, weight decay is typically applied as open-loop control: a fixed coefficient is specified without observing that state. Consequently, an appropriate coefficient must be searched for each task, model, optimizer, and training budget. In grokking in particular, its strength substantially affects both the onset and the stability of delayed generalization.

We propose Thermodynamic Weight Decay (TWD), a dynamic control method inspired by dissipative structures in nonequilibrium thermodynamics. TWD treats the squared weight norm as internal energy, the change induced by the base optimizer as effective energy inflow, and the change induced through weight decay as effective energy dissipation, and controls each block's internal energy toward a target defined from an architecture-aware reference energy.

Our contributions are threefold.

1. **Thermodynamic Weight Decay:** We define an architecture-aware reference energy and a target energy obtained by scaling that reference by a constant factor. Using update information from the base optimizer, TWD solves for the weight-decay coefficient required to make the internal energy track its target. We derive concrete formulations for AdamW and momentum SGD.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

2. **Global vs. block-wise control:** We perform target-energy tracking at the granularity of functional blocks. Compared with global control, which tracks only the total target energy, block-wise control achieves better empirical performance in accelerating grokking and suppressing overfitting.
3. **Around Criticality:** Using TWD's block-wise energy control as a tool that treats criticality as a controllable variable, we broadly sweep the scale factor that maps reference energy to target energy. We show that, along the energy axis, four empirical regimes emerge, including a critical band in which plasticity and stability are balanced. At the same time, these regimes do not share a task-independent common signature in the Jacobian diagnostics, which departs from the conventional view that captures criticality through a single Jacobian condition.

2 Related Work

2.1 From Criticality at Initialization to Control during Training

Mean-field theory showed that deep random networks exhibit a boundary between ordered and chaotic phases, near which signals and gradients propagate effectively (Poole et al., 2016; Schoenholz et al., 2017). Work on dynamical isometry further showed that not only variance but also the singular-value distribution of the input–output Jacobian is important for trainability (Xiao et al., 2018; Fischer et al., 2024), and Zhang et al. (2021) analyzed the relationship between hyperparameters, including weight decay, and the order–chaos phase diagram.

These theories primarily concern activation and gradient propagation and Jacobian properties at initialization. In contrast, we assign architecture-aware reference energies according to the functional roles of parameter blocks and dynamically control their weights during training toward target energies obtained by scaling those references.

2.2 Weight Norms Govern Feature Learning and Grokking

Grokking is the phenomenon in which test performance improves only after training performance has saturated (Power et al., 2022; Liu et al., 2022; Thilak et al., 2022). The formation of periodic internal representations (Nanda et al., 2023) and a relationship between weight norms and generalization (Liu et al., 2023; Kim, 2026) have been reported during this process. Meanwhile, a large output scaling factor can keep training close to the linearization at initialization and induce kernel-like lazy dynamics (Chizat et al., 2019). Kumar et al. (2024) proposed an account of grokking as a transition from lazy dynamics to rich feature learning.

Whereas these studies observe or analyze the relationship between weight scale and feature learning, we directly manipulate the energy level. We measure TWD's relative control magnitude and feature CKA relative to the initial representations to test the relationship between controlled energy levels and rich–lazy dynamics.

2.3 From Existing Norm Control to Target-Tracking Control

Conventional weight decay and AdamW use fixed coefficients (Krogh & Hertz, 1992; Loshchilov & Hutter, 2019), whereas AdaDecay and OUIDecay adapt decay strength using gradient or activation statistics (Nakamura & Hong, 2019; Fernández-Hernández et al., 2026). LARS and LAMB adjust layer-wise step sizes (You et al., 2017; 2020), while max-norm, CPR, projection-based norm constraints, and Weight Norm Control constrain or adjust post-update weight norms (Srivastava et al., 2014; Franke et al., 2024; Loshchilov, 2023).

In contrast, TWD sets an architecture-aware reference energy for each block from its functional role, obtains a target by scaling that reference, and sequentially solves for the weight-decay coefficient before the update while accounting for the actual update of the base optimizer. Because the coefficient may be negative, TWD can move energy toward the target in either direction. It therefore differs in both controlled quantity and control granularity from methods that adapt decay strength from gradient or activation statistics, project norms after an update, or control a single global norm. We provide quantitative comparisons with AdamWN, OrthoGrad, and post-projection in Appendix C.

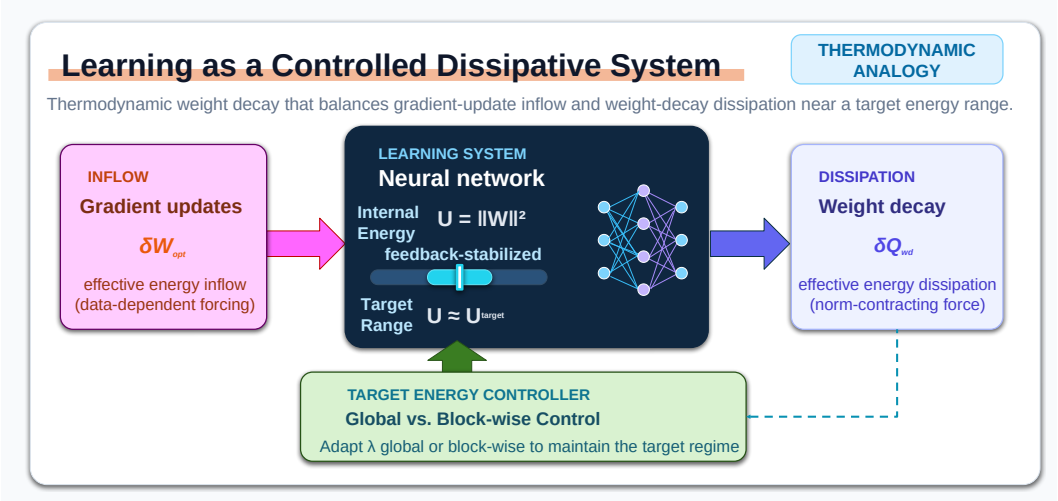

Figure 1: Conceptual diagram of Thermodynamic Weight Decay, inspired by dissipative structures.

3 METHOD

This section formulates the dynamic weight-decay method Thermodynamic Weight Decay (TWD). Figure 1 gives an overview, and the complete pseudocode of the canonical algorithm is provided in Appendix A.1.

3.1 LEARNING DYNAMICS AS A DISSIPATIVE STRUCTURE

In nonequilibrium thermodynamics, a dissipative structure arises in an externally driven open system: order is created through the continual inflow and dissipation of energy, and it is sustained when inflow and dissipation come into balance. Our motivation is to carry this property over to neural-network learning.

Accordingly, we associate the squared weight norm of the network with the internal energy U, the change produced by the base optimizer with the effective inflow $\delta\mathcal{W}_{\mathrm{opt}}$, and the change along the weight-decay path with the controllable dissipation term δQ_{wd}. Corresponding to a discrete first law of thermodynamics, the one-step energy balance reads

$$\Delta U = \delta\mathcal{W}_{\mathrm{opt}} + \delta Q_{\mathrm{wd}}$$

Here, using the weights A_t produced by the base update without weight decay (Section 3.3), the terms are defined as $\delta\mathcal{W}_{\mathrm{opt}} = U(A_t) - U(W_t)$ and $\delta Q_{\mathrm{wd}} = U(W_{t+1}) - U(A_t)$, so that $\Delta U = U(W_{t+1}) - U(W_t)$ holds identically. Each term is signed: a positive weight-decay coefficient ordinarily dissipates energy, whereas a negative coefficient injects energy through the same path.

The nonequilibrium steady state reached after the target is attained corresponds to the equal-energy condition $\Delta U \approx 0$ (equivalently, $\delta\mathcal{W}_{\mathrm{opt}} \approx -\delta Q_{\mathrm{wd}}$). At every step, TWD predicts the energy after the base update and solves for the weight-decay coefficient that satisfies this balance, thereby maintaining the internal energy near a prescribed target energy. It thus operates as closed-loop control that adjusts the amount of dissipation in response to the observed energy state, rather than supplying a fixed coefficient. This thermodynamic correspondence is not asserted as a strict physical identity; rather, it provides a design principle for deriving closed-loop control from the balance between inflow and dissipation.

3.2 REFERENCE ENERGY AND TARGET ENERGY

We partition the model parameters into functional blocks b. Each block comprises several parameters W_i that may play different structural roles r_i; among these, we denote by $\mathcal{C}_b$ the index set of parameters that are assigned a reference energy and controlled by TWD. We define the internal

energy of a block as

$$U_b(W_b) = \sum_{i \in \mathcal{C}_b} \|W_i\|_F^2$$

To each parameter W_i we assign a reference energy u_i^{ref} determined by the architecture $\mathcal{A}$ and its role r_i, and we define the block reference and target energies as

$$u_i^{\mathrm{ref}} := \mathcal{R}(\mathcal{A}, r_i, \mathrm{shape}(W_i)), \qquad U_b^{\mathrm{ref}} := \sum_{i \in \mathcal{C}_b} u_i^{\mathrm{ref}}, \qquad U_b^{\mathrm{target}} = s\, U_b^{\mathrm{ref}}$$

Here $\mathcal{R}$ is a rule that reflects structural roles such as the activation function, initialization scheme, residual connections, normalization, and input/output projections; the Kaiming criterion for ReLU layers and other standard initialization schemes are instances of it (He et al., 2015). The reference energy is fixed before training from the design principles of the model, rather than being a hyperparameter tuned from training outcomes. For each architecture we fix $\mathcal{R}$ and the set of controlled parameters, and we do not search over the numerical value of the reference energy on the basis of validation performance. Additive parameters such as biases are excluded from control but continue to be trained by the base optimizer.

The scale factor s is defined relative to the reference energy; $s = 1$ corresponds to the reference value determined from the structure. In other words, $s = 1$ amounts to taking the reference energy determined from the structure directly as the target, and does not by itself assert criticality or optimality. When the feedback control succeeds, $U_b / U_b^{\mathrm{target}} \approx 1$, and the energy level relative to the reference becomes $U_b / U_b^{\mathrm{ref}} \approx s$. The task-specific rules, block partitions, and exclusions are given in Appendix A.2. Here, the scale factor s is introduced for the criticality analysis presented later, and, unless stated otherwise, we set $s = 1$.

3.3 Predicting the Post-Base-Update Energy

For block b, we predict the weights after an update by the base optimizer without weight decay from the gradient $g_{b,t}$ and the optimizer state:

$$A_{b,t} := \mathrm{OptStep}_{\lambda=0}\big(W_{b,t}, g_{b,t}; \mathrm{state}_t\big)$$

The current energy and the energy after the no-decay update are

$$U_{b,t}^{\mathrm{now}} = \|W_{b,t}\|_F^2, \qquad U_{b,t}^{\mathrm{nodecay}} = \|A_{b,t}\|_F^2$$

This prediction uses the actual optimizer state, including momentum and moment estimates, and the actual parameter update is performed only once. The prediction formulas for each optimizer are given in Appendix A.3.

3.4 Inferring the Target-Tracking Coefficient via the Actual Update Equation

To adjust the rate at which the current energy tracks the target energy, we define the commanded energy as

$$U_{b,t}^{\mathrm{cmd}} = (1 - \rho)\, U_{b,t}^{\mathrm{now}} + \rho\, U_b^{\mathrm{target}}, \qquad 0 < \rho \leq 1$$

A small ρ relaxes the correction, whereas $\rho = 1$ commands one-step tracking to the target. Unless stated otherwise, we set $\rho = 1$ in our experiments.

We write the actual update of each optimizer as a function of the coefficient $\lambda_{b,t}$,

$$W_{b,t+1}(\lambda_{b,t}) = A_{b,t} + \lambda_{b,t} B_{b,t}$$

where $A_{b,t}$ is the weight after the no-decay update and $B_{b,t}$ is the linear term of the weight-decay path. Explicit expressions for AdamW, momentum SGD, Nesterov, and the first-step momentum buffer are given in Appendix A.3. Requiring the post-update energy to equal the commanded energy, $\|A_{b,t} + \lambda_{b,t} B_{b,t}\|_F^2 = U_{b,t}^{\mathrm{cmd}}$, yields the quadratic equation

$$\|B_{b,t}\|_F^2\, \lambda_{b,t}^2 + 2\langle A_{b,t}, B_{b,t}\rangle\, \lambda_{b,t} + \|A_{b,t}\|_F^2 - U_{b,t}^{\mathrm{cmd}} = 0$$

When the no-decay update already reduces the target error at least as much as the commanded energy does, that is, when $|U_{b,t}^{\mathrm{nodecay}} - U_b^{\mathrm{target}}| \leq |U_{b,t}^{\mathrm{cmd}} - U_b^{\mathrm{target}}|$, we set $\lambda_{b,t} = 0$ by a principle

of minimal intervention. Otherwise, among the real roots within the allowed range we select the one with the smallest $|\lambda_{b,t}|$. The fallback used when no feasible root exists, together with the bounds on the weight-decay coefficient, is described in Appendix A.4.

The coefficient computed in this way is passed to each optimizer as its weight-decay coefficient.

3.5 Global vs. Block-wise Control

Global control pools all controlled parameters and applies one coefficient λ_t to the total target energy

$$U_{\text{global}}^{\text{target}} = \sum_b U_b^{\text{target}}$$

Block-wise control instead computes U_b^{target} and $\lambda_{b,t}$ independently for each block. The total target energy is identical in both; the difference is only whether energy allocation across blocks is controlled.

4 Experiments

4.1 Experimental Setup

We evaluate TWD on five tasks that span different learning phenomena and architectures. On modular addition and multiplication we assess the speed and stability of grokking; on a two-layer feedforward network (FFN) for Fashion-MNIST (Xiao et al., 2017) we assess the suppression of overfitting; and on a ResNet-18 (He et al., 2016) for CIFAR-10 (Krizhevsky & Hinton, 2009) we assess generalization to convolutional networks. In addition, we use pretraining of GPT-2 (124M) (Radford et al., 2019) on OpenWebText (Gokaslan & Cohen, 2019) as a case study that examines applicability and computational cost at this model scale. All experiments except GPT-2 are evaluated over four seeds, and the shaded bands in the figures denote standard deviations. The settings for each task are summarized in Table B.2.

As baselines for the basic-performance comparison, we consider four conditions: no control (weight-decay coefficient $= 0.0$), a tuned static weight decay, global TWD, and block-wise TWD (the proposed method). The tuned static baseline is a deliberately strong oracle that selects the best value from a grid on the same task being evaluated. We define the grokking onset as the epoch at which test accuracy first reaches 0.99. For reasons of space, the results for modular multiplication and Fashion-MNIST are deferred to the appendix.

4.2 Modular Addition (One-layer Transformer) and CIFAR-10 (ResNet-18)

The main result for modular addition is shown in Figure 2. Ordered by onset (four-seed averages), block-wise TWD grokked at 448 epochs, tuned static at 540, global TWD at $9{,}093$, and no control at $26{,}733$. Without task-specific tuning, block-wise TWD attains a grokking speed comparable to the oracle-tuned static baseline, accelerating grokking by a factor of approximately 20 relative to global TWD and 60 relative to no control.

For the total energy ratio U/U^{target}, block-wise TWD keeps the ratio at 1 and suppresses softmax collapse (Prieto et al., 2025). Global TWD also tracks the total target energy, but its feedforward block is depleted to 0.73 times the target while its attention and output blocks grow to 1.59 and 2.25 times the target, respectively. With no control, the total energy exceeds six times the target; consequently, both global TWD and no control exhibit softmax collapse.

To confirm TWD applies to convolutional architectures, we evaluate a ResNet-18 on CIFAR-10 (Figure 3). While the final test accuracy of all four conditions falls within a narrow range (approximately 0.91–0.92), the test loss and generalization gap differ (Table B.7). No control, which attains the highest final test accuracy, shows a sharp test-loss increase later in training. In contrast, the tuned static baseline and block-wise TWD, although slightly lower in final test accuracy, suppress the increase in generalization gap and test loss, which we attribute to overfitting suppression. Thus block-wise TWD performs comparably to the tuned static baseline.

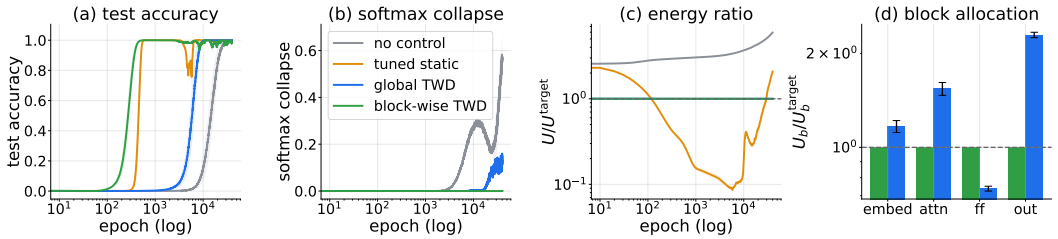

Figure 2: **Main result on modular addition** (40k epochs; mean ± standard deviation over four seeds). (a) Test accuracy; (b) softmax collapse; (c) total energy ratio U/U^{target} (the dashed line in (c) marks the target value of one); (d) final block-wise energy ratio U_b/U_b^{target} (block-wise functional blocks and global; log scale, dashed line at the target of one). Block-wise TWD accelerates grokking by a factor of approximately 60 relative to no control.

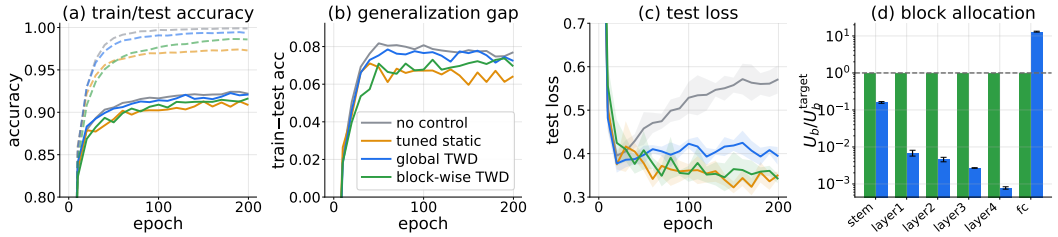

Figure 3: **Main result on CIFAR-10 (ResNet-18)** (s=1; mean ± standard deviation over four seeds; 200 epochs). (a) Train/test accuracy (solid = test, dashed = train); (b) generalization gap; (c) test loss; (d) final block-wise energy ratio U_b/U_b^{target} (block-wise functional blocks and global; log scale, dashed line at the target of one). The final test accuracy is nearly identical across the four conditions (approximately 0.91–0.92), whereas the test loss grows to about 0.57 for no control but is smallest for block-wise TWD and tuned static (about 0.34).

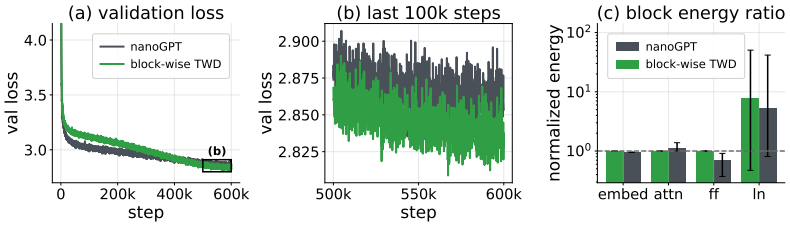

Figure 4: **Pretraining of GPT-2 (124M) on OpenWebText (600k steps).** (a) Validation loss (nanoGPT versus block-wise TWD); (b) magnification of the last 100k steps; (c) layer-averaged normalized block energy for each block group (embedding, attention, feed-forward, and LayerNorm; bars are layer averages, whiskers the layer-wise min–max range). For embedding, attention, and feed-forward we show U_b/U_b^{ref} normalized by the reference energy; because LayerNorm has no reference, we show U_b/U_b^{init} normalized by its initial value. Block-wise TWD attains a validation loss on par with nanoGPT.

4.3 OpenWebText (GPT-2 124M)

Figure 4 reports a case study that examines whether TWD can be implemented for large-scale pretraining, using GPT-2 (124M) and OpenWebText. As a baseline we use the original nanoGPT configuration (Karpathy, 2022). Over the final 10k steps at 600k, the validation loss is 2.862 ± 0.012 for nanoGPT and 2.837 ± 0.011 for block-wise TWD. This experiment does not claim that TWD consistently outperforms nanoGPT; although it is a single-seed case study, it demonstrates that functional-block-wise target tracking can be run for 600k steps in the pretraining of a 124M-parameter model. The computational throughput is reported in Appendix D; the processing time was approximately 1.05 times nanoGPT's.

4.4 Criticality Analysis: Target-Energy Scale Sweep

Sweeping the target-energy scale s, we study how the learning state changes along the energy axis (Figures 5 and 6). Each figure shows (a,b) the learning performance for the task, (c) the feature CKA relative to the initial state, (d) the relative control magnitude of TWD, (e) the energy of the parameters excluded from control, and (f) the largest singular value $\sigma_{\max}$ and the mean squared singular value $\langle \sigma^2 \rangle$ of the Jacobian. Panel (c) reflects representational plasticity, (d) how strongly TWD acts to maintain the target energy, (e) how much energy accumulates in the uncontrolled parameters, and (f) how readily input perturbations are amplified inside the network. We operationally interpret low CKA—a large change from the initial representation—as rich-side feature learning, and high CKA—preservation of the initial representation—as lazy-side learning.

In the low-energy regime for modular addition (Figure 5), CKA is low and the representational change toward the rich side is large (c), while the control magnitude of TWD (d) and the Jacobian signal amplification (f) are also large. For some seeds grokking is faster, but the seed-to-seed variability (a) and the post-onset accuracy drop (b) increase, and at still lower targets some seeds fail to reach grokking (a). In the intermediate regime, sufficient feature learning (c) is retained while the control magnitude (d), the energy of the uncontrolled parameters (e), and the signal amplification (f) become relatively small, and all seeds reach grokking quickly and stably (a,b). In the high-energy regime, CKA rises—chiefly in the embedding and attention blocks—and the dynamics move toward the lazy side (c); the control magnitude decreases (d), whereas the energy of the uncontrolled parameters (e) and the signal amplification (f) increase again, and grokking is delayed or not reached (a). Thus, for modular addition, an intermediate regime that combines speed and stability lies between a low-energy regime in which the change toward the rich side is too strong and becomes unstable and a high-energy regime that remains on the lazy side with insufficient feature learning.

In the low-energy regime for CIFAR-10 (Figure 6), the relative control magnitude of TWD is large (d) and the largest and mean squared singular values of the Jacobian are also large (f), while the test loss is high (b). In the intermediate regime, as the signal amplification (f) and the control magnitude (d) weaken from the low-energy side, the test loss decreases and reaches its minimum near $s \simeq 1$ (b). In the high-energy regime, however, the test loss worsens again (b) even though the control magnitude (d) and the Jacobian diagnostics (f) decrease further. In this regime the feature-CKA response differs across layers (c), and the energy of the uncontrolled parameters increases substantially (e). The generalization performance on CIFAR-10 is therefore not determined monotonically by the Jacobian signal amplification, and it must be interpreted together with the representational change and the energy allocation, including the uncontrolled parameters.

5 Discussion

5.1 Functional-Block-wise Energy Control and Four Empirical Regimes

Our results show that not only the total weight energy of the network but also its allocation across functional blocks is important for learning. The effect of TWD therefore depends on an architecture-aware energy allocation, not merely on constraining the total norm.

Changing the target scale systematically varied control strength, representational change, signal propagation, and learning performance, but their responses and the best scale were task- and architecture-dependent. In the temporal diagnostics, the amplification and anisotropy of the Jacobian formed before the grokking onset, with no discontinuous change around onset. The singular-value analysis in Appendix B.7.3 further showed amplification concentrated onto a few directions with reduced rank on the low-energy side, and amplification across many directions on the high-energy side, so the intermediate regime avoids both extremes. Good learning could thus not be characterized, across all tasks, by a single energy value, by CKA, or by a single signal-propagation criterion.

We therefore regard the finite region that balances the plasticity needed for feature formation with the stability needed to preserve acquired representations as an energy-budget critical band. This working hypothesis is motivated by the Edge of Chaos, the link between weight scale and rich/lazy dynamics, and views of grokking as a lazy-to-rich transition (Poole et al., 2016; Schoenholz et al., 2017; Jacot et al., 2018; Chizat et al., 2019; Kumar et al., 2024). From the sweep of $s = U_b^{\mathrm{target}}/U_b^{\mathrm{ref}}$, we organize the observed behavior into the four empirical regimes of Table 1.

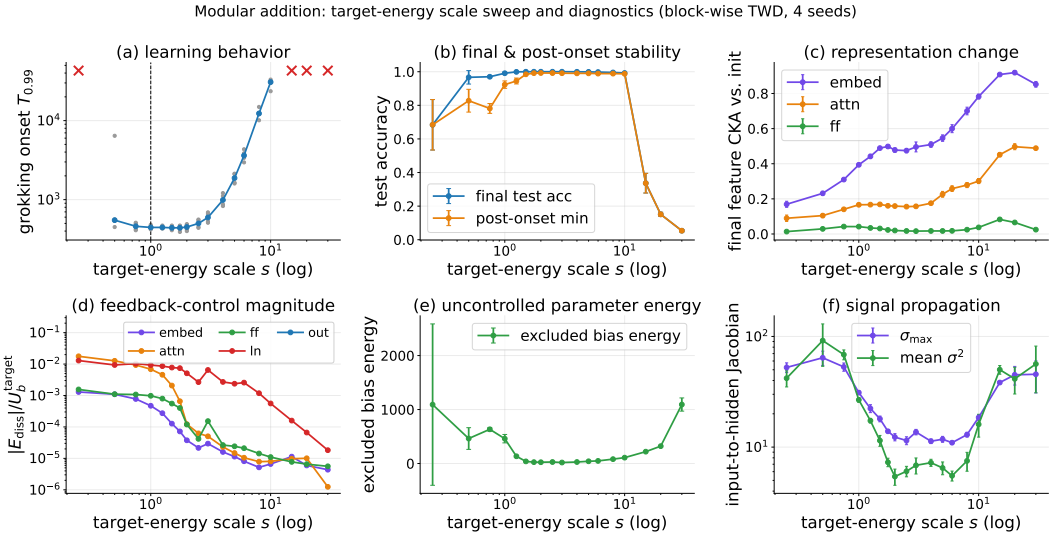

Figure 5: **Target-scale sweep and diagnostics: modular addition** (block-wise TWD; mean $\pm$ standard deviation over four seeds). (a) Grokking onset $T_{0.99}$ (faint points = individual seeds, solid line = median over the 4/4-reached conditions, $\times$ = not reached, with the number reached out of four shown below); (b) final test accuracy and the post-onset minimum test accuracy (post-onset stability); (c) block-wise feature CKA (embedding, attention, feed-forward); (d) block-wise relative control magnitude $|E_{\mathrm{diss}}|/U_b^{\mathrm{target}}$; (e) energy of the excluded parameters (biases); (f) signal propagation (the $\sigma_{\max}$ and mean σ^2 of the input–hidden Jacobian). Under this experimental setting, plasticity and stability are jointly satisfied for roughly $s \simeq 1$–2, and $s = 3$ sits at the boundary of the supercritical side where grokking begins to be delayed; the value of (e) is also low in this region.

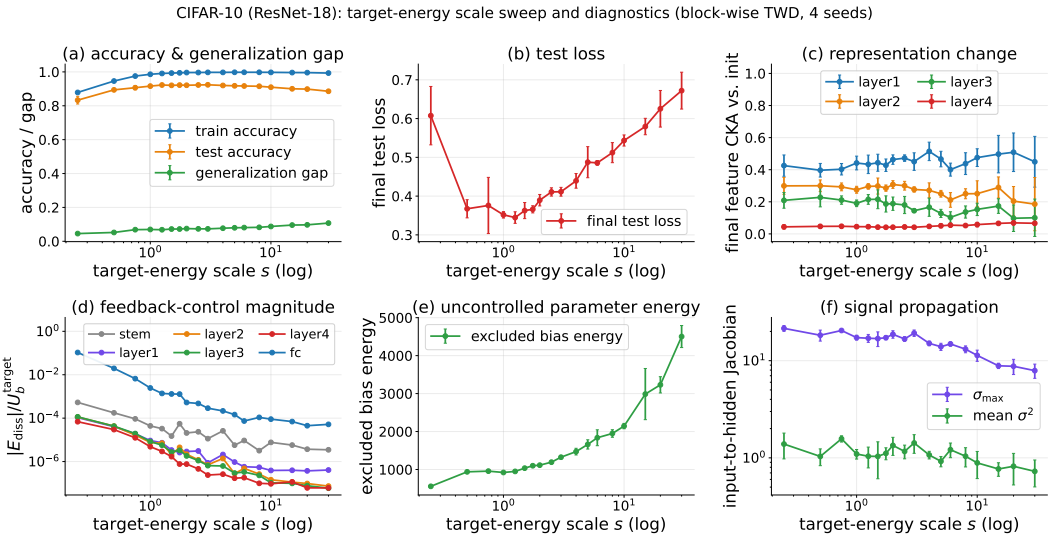

Figure 6: **Target-scale sweep and diagnostics: CIFAR-10 (ResNet-18)** (block-wise TWD; mean $\pm$ standard deviation over four seeds; 200 epochs). (a) Train/test accuracy and generalization gap; (b) final test loss; from (c) onward, the panel layout is identical to Figure 5 (modular addition). Unlike modular addition, the energy of the uncontrolled parameters (e) increases with the scale, whereas the largest and mean squared singular values of the Jacobian (f) mostly decrease. The worsening of the test loss (b) on the high-energy side therefore cannot be explained by Jacobian signal amplification alone.

Table 1: **Four regimes of energy criticality observed in modular arithmetic.** The ranges are empirical values for the architecture-aware block design in our experiments, and the boundaries are continuous. s=3 corresponds to a borderline condition between the critical band and the supercritical regime and does not denote a universal phase boundary. Non-generalization denotes failure to reach within 40k epochs.

Regime	s	Observed behavior	Interpretation
Subcritical	< 1	Increased seed sensitivity and post-onset variability; at still lower values, delay or failure to reach	High plasticity / low stability
Critical band	$\approx$ 1–2	Speed, robustness across seeds, and post-onset stability jointly achieved	Balance of plasticity and stability
Supercritical	$\approx$ 3–8	Stable but delayed feature formation and generalization	Lazy / low plasticity
Non-generalizing regime	$\gtrsim$ 10–15	Remains at memorization within the observation window	Delayed representational change

However, this classification is not a universal phase classification but an organization of the continuous change observed under our experimental conditions. Within it, the critical band is positioned as a dynamic region in which the speed of grokking, robustness across seeds, and the preservation of representations after onset are jointly achieved.

5.2 Limitations and Future Work

Our study has at least the following two limitations.

First, what TWD directly controls is the weight energy, and it does not guarantee that the trained network lies exactly at the Edge of Chaos. Indeed, even in the regime where grokking showed good performance, the largest singular value of the Jacobian was clearly above one, and the full spectrum was not concentrated near one either. Moreover, this response differed across architectures, and no single Jacobian condition could explain all tasks. That is, control on the energy axis does not uniquely determine the signal-propagation properties that lie outside it. In addition, our diagnostics are local measurements at saved checkpoints and do not directly evaluate Lyapunov exponents or correlation propagation at infinite depth, and the full-spectrum analysis is likewise a measurement for a fixed diagnostic input and need not represent the entire input distribution.

Second, although TWD does not require searching for a fixed weight-decay coefficient, its performance depends strongly on the architecture and on the design of the functional blocks and reference energies. We verify its validity through the control-granularity and reference-value comparison experiments (Appendix B.5), but automatically determining the optimal partition of functional blocks and the reference energies is left for future work.

6 Conclusion

This paper is an attempt to adopt a nonequilibrium-thermodynamics frame that treats the squared weight norm as internal energy, and to handle the control mechanism, the criticality sweep, and the diagnostics uniformly on a single axis—energy. Our experiments confirm the effectiveness of the control mechanism, TWD, and lead us to observe four empirical regimes along the energy axis. At the same time, they reveal the difficulty of characterizing the critical band by a single Jacobian condition, showing a good learning regime cannot be captured by a single signal-propagation criterion.

In future work, rather than characterizing the critical band through external Jacobian diagnostics, we aim to formulate the energy-allocation dynamics of TWD as a dynamical system and derive the point at which the linear stability of its stationary allocation bifurcates in s—an analogue of the critical Rayleigh number in dissipative structures. The architecture-aware reference energy we designed in this paper is a first step here.

Reproducibility Statement

The definition and implementation details of the proposed method are provided in Section 3 and Appendix A. The model architectures, optimizers, hyperparameters, seeds, and evaluation conditions for each task are described in Section 4 and Appendix B. Appendix C gives the settings used to compare with existing methods, and Appendix D summarizes the computational cost and the execution environment. The code required for reproduction will be submitted together with the paper as anonymized supplementary material.

Use of Large Language Models

We used large language models (LLMs) to assist in checking consistency between the formulation and the implementation, surveying related work, organizing the experimental design and the interpretation of results, writing experimental, analysis, and visualization code, and refining figures, tables, LaTeX typesetting, and prose. The authors rewrote the LLM-generated drafts of the core control engine and verified and corrected all other generated material. The central idea of Thermodynamic Weight Decay, all research decisions, and the final claims and conclusions are those of the authors, who take responsibility for the entire content of the paper.

References

Lenaic Chizat, Edouard Oyallon, and Francis Bach. On lazy training in differentiable programming. *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Alberto Fernández-Hernández, Jose I. Mestre, Cristian Pérez-Corral, Manuel F. Dolz, Jose Duato, and Enrique S. Quintana-Ortí. OUIDecay: Adaptive layer-wise weight decay for CNNs using online activation patterns. *arXiv preprint arXiv:2605.10161*, 2026.

Kirsten Fischer, Javed Lindner, David Dahmen, Zohar Ringel, Michael Krämer, and Moritz Helias. Critical feature learning in deep neural networks. In *International Conference on Machine Learning (ICML)*, 2024.

Jörg K.H. Franke, Michael Hefenbrock, Gregor Koehler, and Frank Hutter. Improving deep learning optimization through constrained parameter regularization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Aaron Gokaslan and Vanya Cohen. OpenWebText corpus. `http://Skylion007.github.io/OpenWebTextCorpus`, 2019.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *International Conference on Computer Vision (ICCV)*, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (GELUs). *arXiv preprint arXiv:1606.08415*, 2016.

Lei Huang, Xianglong Liu, Bo Lang, and Bo Li. Projection based weight normalization for deep neural networks. *arXiv preprint arXiv:1710.02338*, 2017.

Michael F Hutchinson. A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines. *Communications in Statistics-Simulation and Computation*, 19(2):433–450, 1990.

Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

Andrej Karpathy. nanoGPT. `https://github.com/karpathy/nanoGPT`, 2022. GitHub repository.

Taeyoung Kim. Grokking on the weight-decay clock: A rate hierarchy from softly broken symmetries. *arXiv preprint arXiv:2607.23967*, 2026.

Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In *International Conference on Machine Learning (ICML)*, 2019.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

Anders Krogh and John A. Hertz. A simple weight decay can improve generalization. In *Advances in Neural Information Processing Systems 4 (NeurIPS)*, pp. 950–957, 1992.

Tanishq Kumar, Blake Bordelon, Samuel J. Gershman, and Cengiz Pehlevan. Grokking as the transition from lazy to rich training dynamics. In *International Conference on Learning Representations (ICLR)*, 2024.

Ziming Liu, Ouail Kitouni, Niklas Nolte, Eric J. Michaud, Max Tegmark, and Mike Williams. Towards understanding grokking: An effective theory of representation learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Ziming Liu, Eric J. Michaud, and Max Tegmark. Omnigrok: Grokking beyond algorithmic data. In *International Conference on Learning Representations (ICLR)*, 2023.

Ilya Loshchilov. Weight norm control. *arXiv preprint arXiv:2311.11446*, 2023.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019.

Kensuke Nakamura and Byung-Woo Hong. Adaptive weight decay for deep neural networks. *arXiv preprint arXiv:1907.08931*, 2019.

Neel Nanda, Lawrence Chan, Tom Lieberum, Jess Smith, and Jacob Steinhardt. Progress measures for grokking via mechanistic interpretability. In *International Conference on Learning Representations (ICLR)*, 2023.

Ben Poole, Subhaneil Lahiri, Maithra Raghu, Jascha Sohl-Dickstein, and Surya Ganguli. Exponential expressivity in deep neural networks through transient chaos. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, and Vedant Misra. Grokking: Generalization beyond overfitting on small algorithmic datasets. In *arXiv preprint arXiv:2201.02177*, 2022.

Lucas Prieto, Melih Barsbey, Pedro A.M. Mediano, and Tolga Birdal. Grokking at the edge of numerical stability. In *International Conference on Learning Representations (ICLR)*, 2025.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8), 2019.

Samuel S. Schoenholz, Justin Gilmer, Surya Ganguli, and Jascha Sohl-Dickstein. Deep information propagation. In *International Conference on Learning Representations (ICLR)*, 2017.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

Vimal Thilak, Etai Littwin, Shuangfei Zhai, Omid Saremi, Roni Paiss, and Joshua Susskind. The slingshot mechanism: An empirical study of adaptive optimizers and the grokking phenomenon. In *NeurIPS 2022 Workshop on Optimization for Machine Learning (OPT)*, 2022.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

Lechao Xiao, Yasaman Bahri, Jascha Sohl-Dickstein, Samuel S. Schoenholz, and Jeffrey Pennington. Dynamical isometry and a mean field theory of CNNs: How to train 10,000-layer vanilla convolutional neural networks. In *International Conference on Machine Learning (ICML)*, 2018.

Yang You, Igor Gitman, and Boris Ginsburg. Large batch training of convolutional networks. *arXiv preprint arXiv:1708.03888*, 2017.

Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large batch optimization for deep learning: Training BERT in 76 minutes. In *International Conference on Learning Representations (ICLR)*, 2020.

Lin Zhang, Ling Feng, Kan Chen, and Choy Heng Lai. Edge of chaos as a guiding principle for modern neural network training. *arXiv preprint arXiv:2107.09437*, 2021.

A Algorithm and Reproducibility

A.1 Canonical Algorithm

Figure A.1 gives the canonical procedure and the complete pseudocode of Thermodynamic Weight Decay. At every step, the method measures the current internal energy $U_b^{\mathrm{now}} = \|W_b\|_F^2$ of each block, expresses the actual optimizer update as the affine equation $W_b(\lambda) = A_b + \lambda B_b$, and solves a quadratic equation for the coefficient λ_b that realizes the commanded energy U_b^{cmd}. The principal equations and an overview are given in Sections 3.3 and 3.4; the implementation details of each component are provided in Appendices A.2–A.5.

Require: parameters $\{W_b\}$ grouped into blocks b; learning rate η; feedback rate $\rho \in (0, 1]$ (default 1); coefficient bounds $[\lambda_{\min}, \lambda_{\max}]$; per-block reference energy U_b^{ref}; target energy $U_b^{\mathrm{target}} = s \cdot U_b^{\mathrm{ref}}$

compute gradients g on the current batch
$U_b^{\mathrm{now}} \leftarrow \|W_b\|^2$ for every block b ▷ snapshot current internal energies (all blocks, before any update)
for each block b **do**
  form the affine base step $W_b(\lambda) = A_b + \lambda B_b$ of the actual optimizer ▷ AdamW: $A_b = W_b + \Delta_b^{\mathrm{AdamW}}$, $B_b = -\eta W_b$; SGD analogous
  $U_b^{\mathrm{nodecay}} \leftarrow \|A_b\|^2$ ▷ energy after the base step *without* decay (λ=0)
  $U_b^{\mathrm{cmd}} \leftarrow (1-\rho)\, U_b^{\mathrm{now}} + \rho\, U_b^{\mathrm{target}}$ ▷ commanded energy (ρ=1 $\Rightarrow U_b^{\mathrm{cmd}} = U_b^{\mathrm{target}}$)
  if $|U_b^{\mathrm{nodecay}} - U_b^{\mathrm{target}}| \le |U_b^{\mathrm{cmd}} - U_b^{\mathrm{target}}|$ **then**
    $\lambda_b \leftarrow 0$ ▷ base step already achieves at least the commanded progress: no intervention
  else
    if $\|B_b\| = 0$ **then**
      $\lambda_b \leftarrow 0$
    else
      solve $\|B_b\|^2\lambda^2 + 2\langle A_b, B_b\rangle\lambda + (\|A_b\|^2 - U_b^{\mathrm{cmd}}) = 0$
      if a real root lies in $[\lambda_{\min}, \lambda_{\max}]$ **then**
        $\lambda_b \leftarrow$ such root of smallest $|\lambda|$ ▷ minimal intervention
      else
        $\mathcal{C} \leftarrow \{\lambda_{\min},\ \lambda_{\max},\ \mathrm{clip}(-\langle A_b, B_b\rangle / \|B_b\|^2,\ \lambda_{\min},\ \lambda_{\max})\}$
        $\lambda_b \leftarrow \arg\min_{\lambda\in\mathcal{C}} \left|\|A_b + \lambda B_b\|^2 - U_b^{\mathrm{cmd}}\right|$ ▷ no in-range root: pick the candidate whose energy is closest to the command
      end if
    end if
  end if
  $\lambda_b \leftarrow \mathrm{clip}(\lambda_b,\ \lambda_{\min},\ \lambda_{\max})$ ▷ λ_b may be negative: bidirectional norm control
  $W_b \leftarrow A_b + \lambda_b B_b$ ▷ apply the optimizer step with weight decay λ_b
end for

Figure A.1: **The control law of Thermodynamic Weight Decay.** Using the fact that the actual optimizer update is affine in λ, $W_{\mathrm{next}}(\lambda) = A_b + \lambda B_b$, we solve in each block for the coefficient λ_b that achieves the commanded energy U_b^{cmd}.

A.2 Architecture-aware Functional-Block Design and Reference-Energy Computation

A.2.1 Reference-Energy Construction

In block-wise TWD, parameters are assigned to functional blocks such as embedding, attention, feedforward, normalization, and output projection. Even within the same block, each parameter W_i may play a different structural role r_i. We therefore define a per-parameter reference energy through a rule $\mathcal{R}$ based on the architecture $\mathcal{A}$, the role r_i, and the parameter shape:

$$u_i^{\mathrm{ref}} := \mathcal{R}(\mathcal{A}, r_i, \mathrm{shape}(W_i))$$

Letting $\mathcal{C}_b$ denote the index set of parameters controlled by TWD within block b, the block reference and target energies are

$$U_b^{\mathrm{ref}} = \sum_{i \in \mathcal{C}_b} u_i^{\mathrm{ref}}, \qquad U_b^{\mathrm{target}} = s U_b^{\mathrm{ref}}$$

TWD does not maintain each u_i^{ref} individually; instead, it makes the total energy of a block track the block target $U_b^{\mathrm{target}} = s \sum_{i \in \mathcal{C}_b} u_i^{\mathrm{ref}}$ obtained by summing them. Consequently, within a block the allocation among parameters can be learned while the total energy is maintained.

The rule $\mathcal{R}$ and the set of controlled parameters are not estimated by TWD during training; they are specified before training as a design choice for the architecture. The designer chooses the rule from the activation functions, initialization schemes, residual structure, normalization scheme, and the functional role of each parameter. Once the rule is fixed, each reference energy is computed mechanically from the parameter shape and requires no numerical tuning based on training outcomes. When applying the method to a new architecture, one defines the blocks, roles, reference formulas, and exclusions according to the same principles. This differs from the tuned static baseline, which searches over coefficients or target-energy values based on validation performance. In this work, we fix the rule $\mathcal{R}$ from the architecture and the roles of the parameters and use the default value $s = 1$ in the main comparisons; the sweep of s is carried out only as an analysis of the effect of the energy level.

The basic reference energies used in this work are as follows:

$$u_i^{\mathrm{K,in}} = n_i \frac{2}{f_i^{\mathrm{in}}}, \qquad u_i^{\mathrm{K,out}} = n_i \frac{2}{f_i^{\mathrm{out}}},$$
$$u_i^{\mathrm{Xav}} = n_i \frac{2}{f_i^{\mathrm{in}} + f_i^{\mathrm{out}}}, \quad u_i^{\mathrm{N}}(\sigma) = n_i \sigma^2.$$

Here $n_i = \mathrm{numel}(W_i)$, and f_i^{in} and f_i^{out} are the fan-in and fan-out. Rather than applying these uniformly, we select or correct them according to the activation function, residual connections, normalization scheme, and the input/output role. Parameters that act additively and do not directly determine the signal-amplification factor of a weight matrix, such as biases and the shift of a normalization layer, are excluded from control. Excluded parameters are not frozen: they are updated as usual by the base optimizer, but no TWD coefficient is applied to them. Thus the reference energy is not a physical constant uniquely determined by the architecture, but neither is it a free hyperparameter tuned numerically to validation performance; we treat it as a rule fixed in advance from the function of each parameter and the design principles of the model.

A.2.2 Functional-Block Design: One-layer Transformer for Grokking

For the one-layer Transformer used for modular arithmetic, and to keep the energy allocation across functions separate, we treat the token representation, attention, the feedforward transformation, normalization, and the output projection as separate blocks. The input and output projections within attention, and the two linear layers within the feedforward, each jointly realize a single function and are therefore grouped into the same block.

We thus partition the parameters into the five blocks `embed`, `attn`, `ff`, `ln`, and `out`. The embedding is given a reference energy of 2 per token row, and the attention, feedforward, and output-projection matrices use the Kaiming fan-in reference. The post-norm gains are given a reference of

Table A.1: **Functional blocks and reference energies of the one-layer Transformer** ($p = 113$, $d = 128$, $m = 2048$).

Block	Controlled parameters	Formula	U_b^{ref}
`embed`	`embedding.weight`	$2p$	226
`attn`	`self_attn.in_proj_weight`, `self_attn.out_proj.weight`	$6d + 2d = 8d$	1024
`ff`	`linear1.weight`, `linear2.weight`	$2m + 2d = 2(m + d)$	4352
`ln`	`norm1.weight`, `norm2.weight`	$2 + 2$	4
`out`	`linear.weight`	$2p$	226

2 each, as a Kaiming-type reference for the transformation after normalization rather than the initial energy of a unit gain. All biases are excluded from control.

Table A.1 shows the reference energies for $p = 113$, embedding dimension $d = 128$, and feedforward width $m = 2048$. At $s = 1$, the reference and target energies coincide.

In the attention block,

$$u^{\mathrm{ref}}_{\mathrm{in_proj}} = (3d^2)\frac{2}{d} = 6d, \qquad u^{\mathrm{ref}}_{\mathrm{out_proj}} = d^2\frac{2}{d} = 2d$$

For the feedforward block,

$$u^{\mathrm{ref}}_{\mathrm{linear1}} = 2m, \qquad u^{\mathrm{ref}}_{\mathrm{linear2}} = 2d$$

Because the `out_proj` of `MultiheadAttention` is the output projection of attention, it is included in the `attn` block.

A.2.3 Functional-Block Design: Two-layer FFN for Fashion-MNIST

In the two-layer FFN, the hidden layer that forms the nonlinear feature representation and the output layer that maps this representation to classes play different roles. We therefore control the hidden-layer feature transformation (`hidden`) and the classifier readout (`out`) separately.

For the reference energy of the hidden layer, we use a mean-field reference that reflects the activation function. With hidden width h,

$$u^{\mathrm{ref}}_{\mathrm{hidden}} = n_{\mathrm{hidden}}\frac{g_\phi^2}{f_{\mathrm{in}}} = h\, g_\phi^2, \qquad g_\phi^2 = \begin{cases} 1, & \tanh, \\ 2, & \mathrm{ReLU}. \end{cases}$$

For the tanh network in our experiments, $U^{\mathrm{ref}}_{\mathrm{hidden}} = 784$. For the unnormalized classifier head we use the Xavier reference,

$$U^{\mathrm{ref}}_{\mathrm{out}} = n_{\mathrm{out}}\frac{2}{f^{\mathrm{in}}_{\mathrm{out}} + f^{\mathrm{out}}_{\mathrm{out}}} \simeq 19.75$$

The output bias is excluded from control.

A.2.4 Functional-Block Design: ResNet-18 for CIFAR-10

In ResNet-18, the role of the representation and the energy scale change at each stage where the spatial resolution and the number of channels switch. We therefore treat the stem, the four residual stages, and the final classifier as independent blocks. Within each stage, the main path and the downsample path jointly form a single residual transformation and are grouped into the same block.

We thus partition the parameters into the six blocks `stem`, `layer1`, `layer2`, `layer3`, `layer4`, and `fc`. For the Conv2d weights we use the Kaiming fan-out reference $u_i^{\mathrm{ref}} = n_i \frac{2}{f_i^{\mathrm{out}}}$, corresponding to the ResNet initialization in torchvision. For the BatchNorm gains γ_i we assign $u_i^{\mathrm{ref}} = \mathrm{numel}(\gamma_i)$, corresponding to a unit gain. For the final linear readout we use the expected squared norm of the uniform initialization of PyTorch's `nn.Linear`, $u_i^{\mathrm{ref}} = \frac{n_i}{3f_i^{\mathrm{in}}}$. The convolution and linear biases and the BatchNorm shifts β_i are additive parameters and are excluded from control.

Table A.2: **Functional blocks and reference energies of GPT-2 (124M)** ($V = 50304$, $T = 1024$, $d = 768$, $L = 12$).

Block / parameter	Role	Formula	U_b^{ref}
`embed / wte`	token embedding	$Vd\sigma^2$	15453.3888
`embed / wpe`	position embedding	$Td\sigma^2$	314.5728
`embed` total	shared embedding	$(V+T)d\sigma^2$	15767.9616
`attn.L`ℓ`/c_attn`	attention input proj.	$\frac{2(3d^2)}{d+3d} = \frac{3d}{2}$	1152.0
`attn.L`ℓ`/c_proj`	attention residual proj.	$\frac{1}{2L}\frac{2d^2}{d+d} = \frac{d}{2L}$	32.0
each `attn.L`ℓ total	attention block	$\frac{3d}{2} + \frac{d}{2L}$	1184.0
`ff.L`ℓ`/c_fc`	GELU expansion proj.	$g_{\mathrm{GELU}}^2 \frac{2(4d^2)}{d+4d} = g_{\mathrm{GELU}}^2 \frac{8d}{5}$	2695.6184
`ff.L`ℓ`/c_proj`	FF residual proj.	$\frac{1}{2L}\frac{2(4d^2)}{4d+d} = \frac{4d}{5L}$	51.2
each `ff.L`ℓ total	feedforward block	$g_{\mathrm{GELU}}^2 \frac{8d}{5} + \frac{4d}{5L}$	2746.8184

A.2.5 Functional-Block Design: GPT-2 (124M) for OpenWebText

In nanoGPT, the attention and feedforward of each Transformer layer serve different functions. To avoid distortion of the energy allocation along depth, we control the attention and feedforward of each layer independently as per-layer functional blocks. On the other hand, the input and output projections within the same attention or feedforward jointly realize a single function and are grouped into the same block.

We thus partition the parameters into an `embed` block that shares the token and position embeddings, and `attn.L`ℓ and `ff.L`ℓ blocks for each Transformer layer ℓ. Because the token embedding and the output head are weight-tied, the same parameters are not double-counted.

The embedding uses nanoGPT's initialization standard deviation $\sigma = 0.02$, giving $u_i^{\mathrm{ref}} = u_i^{\mathrm{N}}(0.02) = n_i(0.02)^2$. With L layers, the attention input projection `c_attn` uses the Xavier reference, and the `c_proj` that returns to the residual stream uses a reference that reflects the GPT-2 depth correction:

$$u_{\mathrm{c_attn}}^{\mathrm{ref}} = u_{\mathrm{c_attn}}^{\mathrm{Xav}}, \qquad u_{\mathrm{attn.c_proj}}^{\mathrm{ref}} = \frac{u_{\mathrm{attn.c_proj}}^{\mathrm{Xav}}}{2L}$$

For the feedforward expansion projection c_{fc}, we apply a gain that reflects the gradient propagation of GELU (Hendrycks & Gimpel, 2016). With the exact GELU $\phi(x) = x\Phi(x)$ and $Z \sim \mathcal{N}(0,1)$,

$$\mathbb{E}\big[\phi'(Z)^2\big] \approx 0.455850866, \qquad g_{\mathrm{GELU}}^2 := \frac{1}{\mathbb{E}[\phi'(Z)^2]} \approx 2.1936999$$

Therefore,

$$u_{c_{\mathrm{fc}}}^{\mathrm{ref}} = g_{\mathrm{GELU}}^2\, u_{c_{\mathrm{fc}}}^{\mathrm{Xav}}, \qquad u_{\mathrm{mlp.c_proj}}^{\mathrm{ref}} = \frac{u_{\mathrm{mlp.c_proj}}^{\mathrm{Xav}}}{2L}$$

For GPT-2 (124M), the vocabulary size is $V = 50304$, the maximum sequence length is $T = 1024$, the embedding dimension is $d = 768$, the number of layers is $L = 12$, and the standard deviation of the embedding initialization is $\sigma = 0.02$. Table A.2 shows the reference energies at $s = 1$.

The `ln_1.weight` and `ln_2.weight` of each layer adjust the scale of the representation entering the pre-norm residual branch and may take different values across layers during training, so they are not given a fixed reference energy. The final `ln_f.weight` is also excluded from control because it governs the scale relative to the output head. These LayerNorm gains and all biases are trained as usual by the base optimizer.

A.2.6 Correspondence with Global Control

Global control and block-wise control use the same per-parameter reference energies and the same exclusion set. In global control, all controlled parameters form a single block, and we control

$$U_{\mathrm{global}}^{\mathrm{ref}} = \sum_b U_b^{\mathrm{ref}}, \qquad U_{\mathrm{global}}^{\mathrm{target}} = sU_{\mathrm{global}}^{\mathrm{ref}}$$

The total target energy is therefore the same under both conditions, and the only difference is whether the energy allocation across functional blocks is controlled individually.

A.3 Explicit Prediction Formulas A_b and B_b for Each Base Optimizer

The affine equation $W_b(\lambda) = A_b + \lambda B_b$ is derived from the actual update of the PyTorch optimizer. A_b is the weight after the update without weight decay ($\lambda = 0$), and B_b is the term linear in the weight-decay coefficient.

AdamW (decoupled weight decay): With learning rate η, bias-corrected exponential moving averages of the first and second moments $\hat{m}$ and $\hat{v}$, and a numerical-stabilization term ϵ, one step with weight decay is

$$W_b \leftarrow (1 - \eta\lambda)\, W_b \;-\; \eta\, \frac{\hat{m}}{\sqrt{\hat{v}} + \epsilon}$$

Rearranging this in λ,

$$A_b = W_b - \eta\, \frac{\hat{m}}{\sqrt{\hat{v}} + \epsilon}, \qquad B_b = -\eta\, W_b$$

On the first step, the moving averages of the moments start from zero, and the update is computed as the first Adam step with bias correction $(1 - \beta_1, 1 - \beta_2)$ applied.

Momentum SGD: Let v be the momentum buffer, μ the momentum coefficient, and τ the dampening. In PyTorch SGD, the weight-decay term is added in the gradient direction and acts on the update through the momentum buffer. Decomposing this actual update in λ, the terms A_b and B_b are

$$A_b = W_b - \eta\, d_{\mathrm{opt}}, \qquad B_b = -\eta\, d_{\mathrm{wd}}$$

where d_{opt} is the (momentum-inclusive) direction on the gradient side and d_{wd} is the direction on the weight side. On the first step, when the buffer is uninitialized, $d_{\mathrm{opt}} = g$ and $d_{\mathrm{wd}} = W_b$; thereafter $d_{\mathrm{opt}} = \mu v + g(1 - \tau)$ and $d_{\mathrm{wd}} = W_b(1 - \tau)$ (with an additional momentum term in the Nesterov case). For either optimizer, $\|B_b\| \propto \eta$.

A.4 Other Processing Details

Fallback when no real root exists: We solve the quadratic equation $\|A_b + \lambda B_b\|^2 = U_{b,t}^{\mathrm{cmd}}$ that achieves the commanded energy $U_{b,t}^{\mathrm{cmd}}$. First, if the target error after the no-decay update is no larger than the residual of the command, we set $\lambda_{b,t} = 0$ by the principle of minimal intervention. Otherwise we solve the quadratic equation, and if a real root that realizes the command lies within the interval $[\lambda_{\min}, \lambda_{\max}]$, we select the root of smallest absolute value among them. When no real root lies within the interval—either the real roots are out of range, or the command is below the smallest reachable energy and the discriminant is negative—we evaluate the predicted energy $\|A_b + \lambda B_b\|^2$ at the three candidates, namely the interval endpoints $\lambda_{\min}$ and $\lambda_{\max}$ and the clipped vertex value $\mathrm{clip}(-\langle A_b, B_b\rangle / \|B_b\|^2, \lambda_{\min}, \lambda_{\max})$, and select the one closest to the command. In this way, whether the command is above or below the reachable maximum within the interval, the coefficient closest to the command within the interval is selected.

Range of the weight-decay coefficient: For numerical stability, this implementation restricts the weight-decay coefficient at each step to $\lambda_{b,t} \in [-1/\eta_t, +1/\eta_t]$, where η_t is the learning rate at that step. For AdamW and for ordinary non-Nesterov SGD without dampening, this is equivalent to restricting the direct coefficient $1 - \eta_t \lambda_{b,t}$ on the weight to $[0, 2]$. Setting $\lambda_{b,t} > 1/\eta_t$ on the positive side would cause a sign flip, and setting $\lambda_{b,t} < -1/\eta_t$ on the negative side would amplify the weight direction by more than a factor of two in a single step. In general momentum and Nesterov settings, λ acts through B_b, so we use the same interval as a common numerical safety range in the implementation. This bound applies to the weight-decay path (the coefficient of B_b); because of the base update A_b, even $\lambda_{b,t} = 1/\eta_t$ does not drive the entire weight to zero in a single step.

A.5 Criticality Analysis: Diagnostic Methods in the Target-Energy Scale Sweep

To study how the level of the target energy affects learning performance, feature representations, and signal propagation, we broadly swept the scale factor s of block-wise TWD and analyzed the

learning history and saved checkpoints of each condition. In addition to comparing across scales at the endpoint, for modular addition we used intermediate checkpoints aligned to learning events and measured the temporal change of the signal-propagation properties. This section presents the definitions and aggregation methods for the diagnostics of performance, control magnitude, feature representations, the Jacobian, activations, and gradients. The results are given in Appendix B.

A.5.1 Performance, Relative Control Magnitude, and Feature Representations

For modular addition and multiplication, we defined the grokking onset $T_{0.99}$ as the epoch at which test accuracy first reaches 0.99. Runs that did not reach it within the training budget were treated as right-censored, and we also report the number of seeds that reached it. For Fashion-MNIST and CIFAR-10, we used the final train/test accuracy, the test loss, and the generalization gap.

The relative control magnitude of TWD for block b was measured by normalizing, by the block target energy, the absolute value of the energy E_{diss} that weight decay dissipated from that block at the step:

$$\frac{|E_{\mathrm{diss}}|}{U_b^{\mathrm{target}}}$$

This represents the relative magnitude of the contraction or expansion that TWD applies to the block's weights at that step. Because the correction that matches the initial energy to the target on the first step can be larger than during ordinary training, we excluded the first step and computed, for each run, the average over the recorded steps in the later half of training. We then computed the mean and standard deviation over the four seeds belonging to the same s. For excluded parameters such as biases, we did not define a ratio to the target and instead recorded their squared norm

$$U_{\mathrm{excl}} = \sum_{i \in \mathcal{E}} \frac{1}{2} \|W_i\|_F^2$$

where $\mathcal{E}$ is the set of parameters excluded from TWD control.

The change of the feature representation was measured by linear CKA (Kornblith et al., 2019). For a fixed diagnostic input, letting $H_b^{(0)}$ be the activations of block b obtained from the initial checkpoint and $H_b^{(t)}$ the activations of the evaluated checkpoint,

$$\mathrm{CKA}\left(H_b^{(0)}, H_b^{(t)}\right) = \frac{\left\| \tilde{H}_b^{(0)\top} \tilde{H}_b^{(t)} \right\|_F^2}{\left\| \tilde{H}_b^{(0)\top} \tilde{H}_b^{(0)} \right\|_F \left\| \tilde{H}_b^{(t)\top} \tilde{H}_b^{(t)} \right\|_F}$$

where $\tilde{H}$ denotes the activations centered along the sample direction. A higher CKA indicates that the learned representation is closer to the initial representation. For each run we computed the CKA of the final checkpoint and computed the mean and standard deviation across the seeds belonging to the same s.

A.5.2 Jacobian and Signal-Propagation Quantities

For modular addition and multiplication, the derivative with respect to the discrete tokens themselves is not defined, so we used the continuous representation after the embedding as the input. For an input token sequence x, we set

$$h_{\mathrm{emb}}(x) = \mathrm{Embedding}(x), \qquad h_{\mathrm{tr}}(x) = \mathrm{Transformer}(h_{\mathrm{emb}}(x))$$

and measured the following Jacobian:

$$J(x) = \frac{\partial h_{\mathrm{tr}}(x)}{\partial h_{\mathrm{emb}}(x)}$$

Thus the input is the continuous representation after the embedding, and the output is the Transformer hidden representation just before it enters the final linear classifier. Attention, feed-forward, residual connections, LayerNorm, and the nonlinear activations are all included in this map.

For Fashion-MNIST, letting x be the flattened input image, $W_1x + b_1$ the first linear layer, ϕ the activation function, and W_2 the output layer, we measured

$$J(x) = \frac{\partial\, W_2\, \phi(W_1x + b_1)}{\partial x}$$

that is, the Jacobian from the input image through the full two-layer network to the output logits. Because this Jacobian is a 10×784 rectangular matrix, we used $d = \min(d_{\text{in}}, d_{\text{out}}) = 10$ as the normalization dimension of the mean squared singular value.

For CIFAR-10, we measured the Jacobian from the input image $x \in \mathbb{R}^{3\times32\times32}$ to the 512-dimensional representation $h_{\text{pool}}(x)$ after the global average pooling of ResNet-18 and before the final classification layer:

$$J(x) = \frac{\partial h_{\text{pool}}(x)}{\partial x}$$

This map includes the stem, the four residual stages, BatchNorm, ReLU, the skip connections, and the global average pooling, but not the final linear classification layer. This diagnostic therefore evaluates the signal propagation from the input image to the feature representation just before the classifier, not the sensitivity to the class logits. Because the Jacobian is a 512×3072 rectangular matrix, we used

$$d = \min(d_{\text{in}}, d_{\text{out}}) = 512$$

as the normalization dimension of the mean squared singular value.

The largest singular value $\sigma_{\max}(J)$ was estimated by power iteration using Jacobian-vector products and vector-Jacobian products, without constructing the Jacobian explicitly. We also defined the mean squared singular value as

$$\langle \sigma^2 \rangle = \frac{1}{d}\|J\|_F^2$$

and estimated it by a Hutchinson estimator (Hutchinson, 1990) using Rademacher vectors, where d is the number of singular values of the Jacobian. For the endpoint scale diagnostics, we evaluated 8 samples from a fixed diagnostic batch of 32 samples, using 15 power iterations and 16 Hutchinson probes. We first computed the average over samples within each checkpoint and then the mean and standard deviation over the four seeds belonging to the same s.

In addition, at the main computational stages of each network, we recorded the activation variance

$$V_\ell = \frac{1}{N_\ell}\sum_{j=1}^{N_\ell}\left(h_{\ell,j} - \bar{h}_\ell\right)^2$$

and the gradient RMS

$$G_\ell = \sqrt{\frac{1}{N_\ell}\left\|\frac{\partial \mathcal{L}}{\partial h_\ell}\right\|_2^2}$$

where h_ℓ is the activation at stage ℓ and N_ℓ is its number of elements. These were used to complementarily evaluate the amplification or attenuation of signals and gradients inside the network.

A.5.3 Temporal Diagnostics of Signal Propagation

To study not only the final state but also how the signal-propagation properties form during training, for modular addition we analyzed intermediate checkpoints over all 18 scales

$$s \in \{0.25, 0.5, 0.75, 1, 1.25, 1.5, 1.75, 2, 2.5, 3, 4, 5, 6, 8, 10, 15, 20, 30\}$$

and four seeds. From each run, we selected the checkpoints corresponding to the following learning events:

1. the saved checkpoint at epoch 0;
2. the point at which train accuracy first became at least 0.99;
3. just before the grokking onset;
4. just after the grokking onset;

5. the final checkpoint.

Because checkpoints were saved at intervals of 10 epochs, the measurement points just before and just after onset are in principle 10 epochs apart. For runs that did not reach train accuracy 0.99 or the grokking onset within the observation period, the corresponding events were treated as missing. This procedure selected 319 checkpoints in total from 72 runs.

For the temporal diagnostics, we evaluated the Jacobian for 16 samples from a fixed batch of 64 samples, using 20 power iterations and 16 Hutchinson probes. For each learning event, we first computed the average over diagnostic samples within a checkpoint, and then the mean and standard deviation across the seeds belonging to the same s. For events that are defined only for reached runs, we also report the number of valid seeds for each s. Because the epoch-0 checkpoint represents the first save point in the training code, we distinguished it from the exact pre-update initial values.

This event alignment allows us to compare the Jacobian, activation variance, and gradient RMS at the memorization of the training data, at the grokking onset, and at the final state, in correspondence across scales whose absolute learning times differ greatly.

A.5.4 Singular-Value Distribution

The largest singular value and the mean squared singular value alone cannot distinguish a concentration onto a few large singular values from a collapse of many singular values toward zero. We therefore explicitly constructed the 256×256 Jacobian for a fixed single diagnostic sample, for all 319 checkpoints used in the temporal diagnostics, and obtained all singular values $\{\sigma_i\}_{i=1}^{256}$ by a full SVD

$$J = Q\Sigma V^\top$$

We summarized the resulting distribution, in addition to the maximum and the mean square, by the median, the quantiles, and the effective rank

$$r_{\text{eff}} = \exp\left(-\sum_i p_i \log p_i\right), \qquad p_i = \frac{\sigma_i^2}{\sum_j \sigma_j^2}$$

For each scale and learning event, we first summarized the singular-value distribution of each seed by quantiles and the effective rank, and then computed the mean and standard deviation across the reached seeds. This allowed us to assess whether the anisotropy of the Jacobian and the effective propagation dimension differ even between conditions with similar estimated $\sigma_{\max}$ or $\langle\sigma^2\rangle$.

For the 319 checkpoints used in the temporal diagnostics, a comparison with the full SVD showed that the power-iteration estimate of the largest singular value had a median relative error of 0.004% and the Hutchinson estimate of the mean squared singular value had 4.3%, indicating sufficient accuracy for scale comparison. What TWD directly controls is the weight energy of each functional block, not the singular values of the Jacobian. These diagnostics therefore do not assume that dynamical isometry holds; they are intended to evaluate the empirical relationship between the target energy and the signal-propagation properties during training.

B Detailed Experimental Results

This appendix presents experimental results that could not be included in Section 4. First, Table B.1 summarizes the correspondence between the main results and the figures/tables. The experimental settings for each task are summarized in Table B.2.

Table B.1: **Correspondence between the main results and the scope of validation.** We show the validation targets (architecture and learning phenomenon) and the figures/tables that support each result. All experiments except the GPT-2 pretraining are four-seed averages, and GPT-2 is a single-seed case study. Details such as the model and the training budget are given in Table B.2.

Main result	Validation target	Figures/tables
Acceleration of grokking	Addition, multiplication	Figures 2, B.2
Comparison of global and block-wise control	Addition, multiplication, Fashion-MNIST, CIFAR-10	Figures 2, 3, B.2, B.3
Suppression of overfitting	Fashion-MNIST, CIFAR-10	Figures 3, B.3
Energy level and learning dynamics	Addition, multiplication, Fashion-MNIST, CIFAR-10	Figures 5, 6, B.6, B.7, B.8, B.9, B.10
Applicability at the 124M scale	GPT-2 / OpenWebText (1 seed)	Figure 4, Table B.9, Figure D.2

Table B.2: **Principal experimental settings per task.**

Setting	Addition/Multiplication	Fashion-MNIST	CIFAR-10	GPT-2 pretraining
Model	One-layer Transformer	Two-layer FFN (hidden 784, tanh)	ResNet-18	GPT-2 124M*
Data	$(a \circ b) \bmod p$ (p=113)	784-dim $[0, 1]$ flattened	CIFAR-10 (standard aug.)	OpenWebText
Train split	training fraction 0.4	standard split (no aug.)	standard train/test split	—
Loss	cross-entropy	cross-entropy	cross-entropy	cross-entropy
Base optimizer	AdamW	momentum SGD	momentum SGD	AdamW
Learning rate	10^{-3}	0.02	0.1	6×10^{-4}
Optimizer-specific	β=$(0.9, 0.999)$	momentum=0.5	momentum=0.9	β=$(0.9, 0.95)$
Weight decay	controlled by TWD	controlled by TWD	controlled by TWD	0.1 (base) / controlled by TWD
Gradient clipping	—	—	—	1.0
Batch size	full batch	32	128	491,520 tok/step
Training budget (default/ext.)	40k / 100k epochs	500 / 1,500 epochs	200 epochs	600k steps
Seed	0–3	0–3	0–3	1337

* The implementation follows nanoGPT (Karpathy, 2022): we use the Transformer model defined in nanoGPT and follow its learning-rate scheduler, gradient scaler, and evaluation-loop structure.

B.1 Selection of the Tuned Static Weight Decay

The oracle-tuned static baseline was searched on the same setting as the evaluation target. For modular arithmetic, among the candidates that maintained the final test accuracy, we selected the value with the smallest onset, and adopted $wd = 5$ for both addition and multiplication (Table B.3). For Fashion-MNIST, based on the test accuracy after 500 epochs, we adopted $wd = 3 \times 10^{-4}$ (Table B.4). For CIFAR-10, we adopted $wd = 10^{-4}$ (Table B.5). These are comparison conditions favorable to a fixed weight decay chosen using the evaluation results, whereas TWD uses the common $s = 1$ for all tasks.

Table B.3: **Selection of the tuned static baseline for modular addition and multiplication (4 seeds, 40k epochs).** Among the candidates that maintained the final test accuracy, we selected the weight decay with the smallest onset. The star marks the adopted value (wd=5 for both addition and multiplication). Onset and final acc are averages over four seeds. The display is thinned to representative values. NR indicates that only some of the four seeds reached (including failure to reach within budget).

wd	0	1	2	3	4	5$\star$	6	8	10
Addition onset	26,733	1,712	998	740	612	540	485	**452**	NR
Addition final acc	0.999	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	0.979	0.394	0.180
Multiplication onset	21,410	1,528	878	648	538	478	428	**390**	NR
Multiplication final acc	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	0.758	0.348	0.177

Onset is the average over four seeds. For wd=10, only 3 of the 4 seeds reached for both addition and multiplication and the final test accuracy collapses, so it is marked NR and the onset is not aggregated. The final test accuracy collapses for $wd \geq 6$.

Table B.4: **Selection of the tuned static baseline for Fashion-MNIST (4 seeds, 500 epochs).** Sweeping the fixed weight decay, the adopted value ($\star$) is chosen by the following criterion: among the neighborhood where the max test accuracy is largest, the one with the smallest final test loss. In this data, the max test accuracy is largest at wd=10^{-4} (0.896), but the smallest final test loss in that neighborhood is at wd=3×10^{-4} (0.322), which we adopted.

wd	0	10^{-4}	3×10^{-4} $\star$	10^{-3}	3×10^{-3}	10^{-2}	3×10^{-2}
final train acc	**1.000**	0.989	0.953	0.903	0.872	0.838	0.804
final test acc	0.890	**0.892**	0.890	0.877	0.855	0.825	0.786
max test acc	0.892	**0.896**	0.895	0.882	0.860	0.836	0.801
final test loss	0.705	0.385	**0.322**	0.343	0.410	0.512	0.654
generalization gap	0.110	0.096	0.064	0.027	0.017	**0.014**	0.018

Table B.5: **Selection of the tuned static baseline for CIFAR-10 (4 seeds, 200 epochs).** Sweeping the fixed weight decay, we adopted the value with the smallest final test loss among the positive weight decays. wd=10^{-4} has the smallest final test loss (0.350) and also the highest final and maximum test accuracy among the positive weight decays, so we adopted it as the tuned static baseline. Note that wd=0 is treated separately as the no-control comparison condition.

wd	0	10^{-4} $\star$	3×10^{-4}	5×10^{-4}	10^{-3}	3×10^{-3}	10^{-2}	3×10^{-2}
final train acc	**0.999**	0.973	0.930	0.900	0.844	0.709	0.451	0.143
final test acc	**0.922**	0.909	0.877	0.837	0.786	0.530	0.226	0.122
max test acc	**0.925**	0.916	0.900	0.879	0.816	0.673	0.385	0.166
final test loss	0.570	**0.350**	0.383	0.505	0.637	1.679	2.592	2.331
generalization gap	0.077	0.064	0.053	0.063	0.057	0.179	0.225	**0.021**

B.2 Control Operation Statistics (Fallback, Clipping, Quadratic Inversion)

The TWD coefficient $\lambda_{b,t}$ is obtained by solving the quadratic equation that returns each block's energy to the target. To confirm that this quadratic inversion works stably as the normal path, we aggregated the control decisions for each (block, step) from the training logs at $s = 1$ and seed 0 (Table B.6). As representatives we use modular addition (10k epochs) and CIFAR-10 (200 epochs), the two tasks presented in the main text.

Table B.6: **Operation statistics of the TWD controller (s=1, seed 0).** Each (block, step) is aggregated as one control decision. The exact-root rate $= N_{\text{exact root}}/N_{\text{all control decisions}}$ (the denominator being the number of all control decisions) is the fraction of decisions that fall into none of fallback (no real root, recorded in `skip_reasons`), clipping (candidate coefficient $\neq$ applied coefficient), or zero-intervention (λ=0). "track. err." is $\text{median}_t\, |U_b/U_b^{\text{target}} - 1|$, and $\eta|\lambda|$ is $\text{median}_t\, \eta_t|\lambda_{b,t}|$. Blocks are ranked by the median of $\eta|\lambda|$, and we show maximum-control (largest) and median-control (middle rank). For modular addition, to reduce computation we aggregated the first 10k epochs as the analysis interval; for CIFAR-10 we used the entire 200 epochs.

task	block	exact root	zero int	fallback	clip	neg λ	track err	$\eta\|\lambda\|$ (med)
modular add	Overall	99.994%	0.0%	0.0%	0.006%	0.06%	3.0×10^{-8}	2.8×10^{-3}
	max-ctrl (attn)	99.99%	0.0%	0.0%	0.01%	0.18%	3.7×10^{-8}	3.8×10^{-3}
	med-ctrl (ln)	100.0%	0.0%	0.0%	0.0%	0.04%	4.3×10^{-8}	2.8×10^{-3}
CIFAR-10	Overall	100.0%	0.0%	0.0%	0.0%	15.8%	2.3×10^{-8}	7.2×10^{-6}
	max-ctrl (fc)	100.0%	0.0%	0.0%	0.0%	21.0%	4.8×10^{-8}	1.8×10^{-3}
	med-ctrl (layer2)	100.0%	0.0%	0.0%	0.0%	0.0%	1.6×10^{-8}	6.0×10^{-6}

Across all blocks of the two aggregated tasks, no fallback due to an infeasible root (a negative discriminant) ever occurred, and zero-intervention (λ=0) was also 0%. Clipping (the candidate coefficient falling outside the interval $[\lambda_{\min}, \lambda_{\max}]$ and being clamped to a bound) occurred only at the first step (step 0) of modular addition, when the initial energy is corrected in one shot, for the three blocks attn, ff, and out; at every other step the real root of the quadratic inversion was adopted directly. Consequently the exact-root rate was 99.994% (clip 0.006%) for modular addition and 100% for CIFAR-10, showing that the quadratic inversion operates stably as the normal path. The tracking error (the median of $|U_b/U_b^{\text{target}} - 1|$) was extremely small at $\sim 10^{-8}$, so the target tracking is accurate.

On the other hand, the negative-λ rate ($\lambda < 0$, i.e., energy injection) differed greatly across tasks and blocks. It was almost 0% for modular addition but reached 15.8% overall for CIFAR-10. While the classifier block (fc) required the largest coefficients, negative coefficients occurred most frequently in the stem (35.5%) and layer1 (32.0%), indicating that these early feature-extraction blocks repeatedly required energy injection rather than dissipation. Bidirectional control (both dissipation and injection) is therefore not a formal feature but is used in actual training. Figure B.1 shows the temporal change of $\lambda_{b,t}$ for each block. For addition, after an initial transient injection the coefficients converge to a positive steady value, whereas for CIFAR-10 the stem, layer1, and fc repeatedly flip sign throughout training.

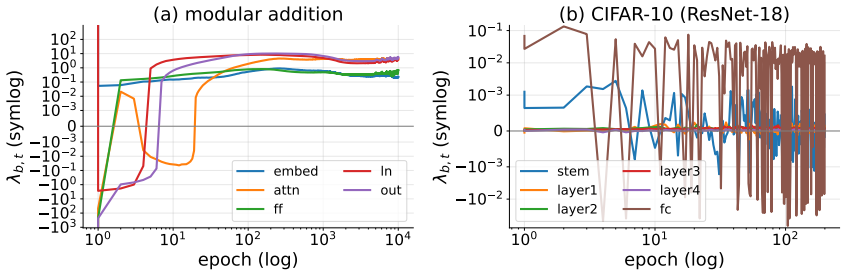

Figure B.1: **Temporal change of the block-wise dynamic weight decay $\lambda_{b,t}$ on modular addition and CIFAR-10 (ResNet-18)** (s=1, seed 0; these two runs are the representative results shown in the main text; the uncontrolled excluded parameters such as biases are not shown). (a) Modular addition (log epoch); (b) CIFAR-10 (ResNet-18). The vertical axis is symlog (sign-preserving). Negative coefficients (energy injection) appear in the initial transient; for addition all blocks converge to a positive steady value, whereas for CIFAR-10 the stem, layer1, and fc frequently flip sign throughout training, showing that bidirectional control is used continuously.

B.3 Basic Performance: Modular Multiplication (One-layer Transformer) and Fashion-MNIST (Two-layer FFN)

The main result for modular multiplication is shown in Figure B.2. Ordered by onset (all four-seed averages), we obtain block-wise TWD at 370, tuned static at 478, global TWD at 7,555, and no control at 21,410 epochs. Without searching for a task-specific fixed weight-decay coefficient, block-wise TWD attains a grokking speed comparable to the oracle-tuned static baseline, accelerating grokking by a factor of approximately 20 relative to global TWD and by a factor of approximately 58 relative to no control.

For the total energy ratio U/U^{target}, block-wise TWD keeps the ratio at 1 and suppresses softmax collapse (Prieto et al., 2025). Global TWD also tracks the total target energy, but its feedforward block is depleted to 0.74 times the target while its attention and output blocks grow to 1.50 and 2.36 times, respectively. With no control, the total energy grows substantially, and consequently both global TWD and no control exhibit softmax collapse.

The main result for Fashion-MNIST is shown in Figure B.3. The final test accuracy is nearly identical at about 0.89, but a difference appears in the generalization gap and the test loss. No control overfits, with the gap growing to 0.110 and the test loss to 0.71, whereas tuned static and block-wise TWD keep the test loss at about 0.32. In the block-wise energy ratio (Figure B.3(d)), global control maintains the total energy but depletes the hidden layer to about 0.70 times the target and inflates

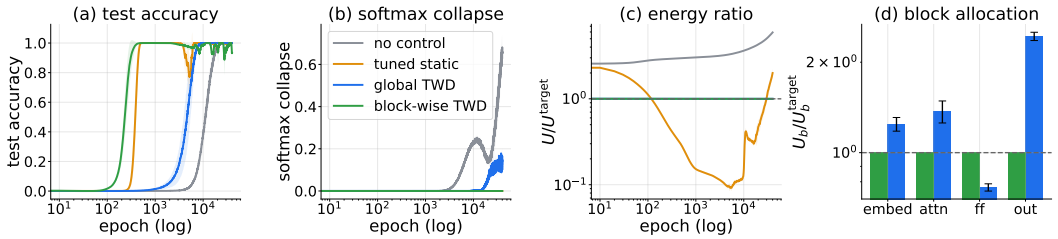

Figure B.2: **Main result on modular multiplication** (40k epochs; mean ± standard deviation over four seeds). (a) Test accuracy; (b) softmax collapse; (c) total energy ratio U/U^{target} (the dashed line in (c) marks the target value of one); (d) final block-wise energy ratio U_b/U_b^{target} (block-wise functional blocks and global; log scale, dashed line at the target of one). Block-wise TWD accelerates grokking by a factor of approximately 58 relative to no control.

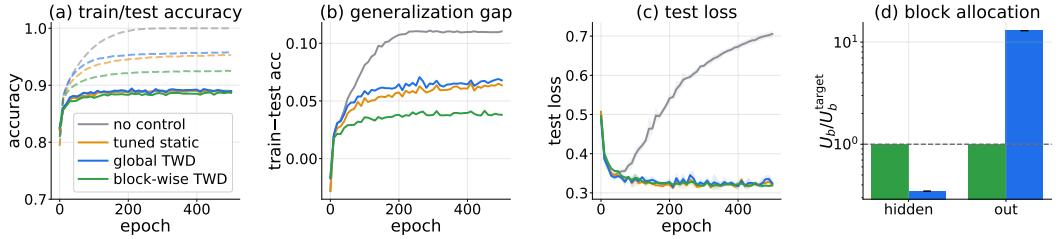

Figure B.3: **Main result on Fashion-MNIST (Two-layer FFN)** (mean ± standard deviation over four seeds; 500 epochs). (a) Train/test accuracy (solid = test, dashed = train); (b) generalization gap; (c) test loss; (d) final block-wise energy ratio U_b/U_b^{target} (block-wise functional blocks and global; log scale, dashed line at the target of one). No control overfits and its test loss grows, whereas the tuned static and TWD conditions suppress the test loss. Global control maintains the total energy but depletes the hidden layer and inflates the output layer, whereas block-wise control keeps each block at the target.

the output layer to about 13 times, whereas block-wise control keeps both blocks at the target (about 1.00). Thus the allocation error of global control also occurs on Fashion-MNIST, and functional block-wise control is effective.

B.4 Basic Performance: CIFAR-10 (ResNet-18), Supplementary

For the four conditions of CIFAR-10 (ResNet-18) in Section 4.2 and Figure 3, Table B.7 shows the final-checkpoint values (200 epochs, four seeds, mean ± std).

Table B.7: **Final metrics of the four conditions on CIFAR-10 (ResNet-18)** (200 epochs, four seeds, mean ± std). "final acc"/"max acc" are the test accuracy at the final epoch / the maximum during training, and "gap" is the final train−test accuracy. The final test accuracy falls within a narrow range across the four conditions, whereas the TWD conditions and tuned static substantially reduce the test loss.

method	final acc	max acc	test loss	gap
no control	**0.922±0.006**	**0.925±0.005**	0.570±0.032	0.077±0.006
tuned static	0.909±0.002	0.916±0.001	0.350±0.009	**0.064±0.000**
global TWD	0.921±0.004	**0.925±0.003**	0.395±0.020	0.073±0.004
block-wise TWD	0.916±0.001	0.918±0.001	**0.343±0.011**	0.070±0.002

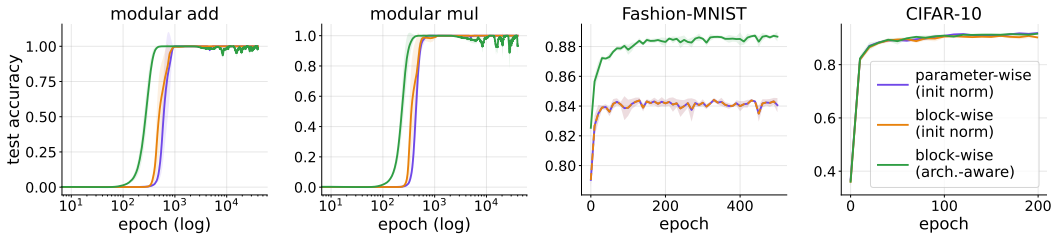

Figure B.4: **Effect of control granularity and reference energy** (s=1; mean ± standard deviation over four seeds). Learning curves of the test accuracy for the three variants (parameter-wise, init norm / block-wise, init norm / block-wise, architecture-aware) are shown for (a) modular addition, (b) modular multiplication (log epoch on the horizontal axis), (c) Fashion-MNIST, and (d) CIFAR-10 (in (c) and (d) the vertical axis is zoomed to the convergence region). On grokking both changes shorten the onset; on Fashion-MNIST the reference (init norm → architecture-aware) is dominant, and on CIFAR-10 the granularity (parameter-wise → block-wise) is dominant.

Table B.8: **Effect of control granularity and reference energy** (s=1, four seeds, mean ± std). For grokking we show the onset $T_{0.99}$ (epochs; smaller is faster); for Fashion-MNIST and CIFAR-10 we show the final test accuracy, test loss, and generalization gap for reference. The best value in each row is shown in bold. The generalization gap is a reference value and is shown in parentheses.

arch.	metric	parameter-wise (init norm)	block-wise (init norm)	block-wise (arch.-aware)
Transformer	modular add: onset	890±88	800±106	**448±22**
	modular mul: onset	642±280	597±78	**370±47**
FFN	Fashion-MNIST: acc	0.841±0.005	0.841±0.005	**0.887±0.003**
	Fashion-MNIST: test loss	0.459±0.008	0.459±0.008	**0.319±0.006**
	(Fashion-MNIST: gap)	(0.016±0.005)	(0.016±0.005)	(0.038±0.003)
ResNet-18	CIFAR-10: acc	0.902±0.003	**0.918±0.005**	0.916±0.001
	CIFAR-10: test loss	0.371±0.017	**0.339±0.016**	0.343±0.011
	(CIFAR-10: gap)	(0.071±0.003)	(0.068±0.004)	(0.070±0.002)

B.5 Basic Performance (additional): Effect of Control Granularity and Reference Energy

To disentangle whether the effectiveness of block-wise TWD derives from (i) the control granularity or (ii) how the reference energy is defined (init-norm vs. architecture-aware), we compare parameter-wise TWD (init-norm reference) and block-wise TWD (init-norm reference) against the proposed block-wise TWD (architecture-aware) (Figure B.4; values in Table B.8).

Overall, block granularity is preferable, but the dominant factor differs by task. On grokking, block-wise TWD (architecture-aware) greatly shortened the onset (448 for addition, 370 for multiplication). On Fashion-MNIST, granularity had no effect under the init-norm reference (0.841 for both), whereas the architecture-aware reference improved accuracy to 0.887. Conversely, on CIFAR-10 the reference mattered little (init norm 0.918 vs. architecture-aware 0.916), while moving from per-parameter to per-block granularity improved accuracy from 0.902 to about 0.92.

Thus, for granularity, function-grouped block-wise control is more effective than per-parameter or global control, whereas the reference-energy setting is task- and architecture-dependent and leaves room for improvement—for example, on CIFAR-10 the init-norm reference performed best, suggesting an appropriate initialization.

B.6 Basic Performance: OpenWebText (GPT-2 124M), Supplementary

Table B.9 shows the per-block internal energy of block-wise TWD and nanoGPT at the final checkpoint.

At 600k steps, block-wise TWD maintained all controlled blocks at their prescribed reference energies. Within the blocks, however, a substantial energy redistribution occurred, from the attention input projection to the output projection and from the feed-forward expansion projection to the residual projection (Table B.9b). This shows that block-wise TWD, while constraining the total energy of each functional block, preserves within it the degrees of freedom needed for learning.

Table B.9: **Final per-block internal energy of GPT-2 (124M) (s=1.0, 600k steps).** The initial values are the weight energy at epoch 0, and the uncontrolled nanoGPT is the value at the final checkpoint under the same training setting.

(a) Per-block internal energy. Block-wise TWD maintains each block's total energy at the reference value while allowing redistribution among the parameter groups within a block; the LayerNorm gains are uncontrolled and have no reference value ($\ln .\mathrm{L}xx$ is $\ln_1 + \ln_2$).

Block	Initial	Reference U_b^{ref}	block-wise TWD	nanoGPT
Controlled				
embed	15,765	15,768	15,768	15,003
attn.L00	717	1,184	1,184	1,635
attn.L01	717	1,184	1,184	1,507
attn.L02	718	1,184	1,184	1,313
attn.L03	717	1,184	1,184	1,360
attn.L04	718	1,184	1,184	1,402
attn.L05	719	1,184	1,184	1,235
attn.L06	718	1,184	1,184	1,275
attn.L07	718	1,184	1,184	1,184
attn.L08	717	1,184	1,184	1,217
attn.L09	717	1,184	1,184	1,248
attn.L10	717	1,184	1,184	1,301
attn.L11	719	1,184	1,184	1,314
ff.L00	983	2,747	2,747	1,785
ff.L01	981	2,747	2,747	1,023
ff.L02	983	2,747	2,747	1,864
ff.L03	983	2,747	2,747	1,913
ff.L04	983	2,747	2,747	2,046
ff.L05	981	2,747	2,747	1,767
ff.L06	983	2,747	2,747	1,829
ff.L07	982	2,747	2,747	1,855
ff.L08	982	2,747	2,747	2,027
ff.L09	984	2,747	2,747	2,209
ff.L10	984	2,747	2,747	2,460
ff.L11	984	2,747	2,747	2,506
Controlled total	**36,172**	**62,938**	**62,938**	**54,291**
Uncontrolled (LayerNorm gains)				
ln.L00	1,536	—	724	1,240
ln.L01	1,536	—	9,752	2,883
ln.L02	1,536	—	6,460	4,238
ln.L03	1,536	—	5,862	3,541
ln.L04	1,536	—	5,444	3,037
ln.L05	1,536	—	5,658	3,313
ln.L06	1,536	—	5,745	3,341
ln.L07	1,536	—	6,389	3,486
ln.L08	1,536	—	6,385	3,926
ln.L09	1,536	—	7,366	3,670
ln.L10	1,536	—	8,728	4,091
ln.L11	1,536	—	9,688	4,314
ln_f	768	—	38,751	31,984
Uncontrolled total	**19,200**	—	**116,951**	**73,061**
All-parameter total	**55,372**	—	**179,889**	**127,352**

(b) Parameter groups within the controlled blocks (12-layer average; not individually constrained). Block-wise TWD controls only the total energy of the attention and feed-forward blocks (U_{attn}=1,184 and U_{ff}=2,747, respectively); it maintains the block total while redistributing substantially inside, and in particular the feed-forward residual projection grew to approximately 24 times the reference value.

Parameter group (12-layer avg)	Initial	Reference U_b^{ref}	block-wise TWD	nanoGPT
attn / c_{attn}	708	1,152	986	1,148
attn / c_{proj}	10	32	198	185
ff / c_{fc}	944	2,696	1,514	1,168
ff / mlp.c_proj	39	51	1,232	773

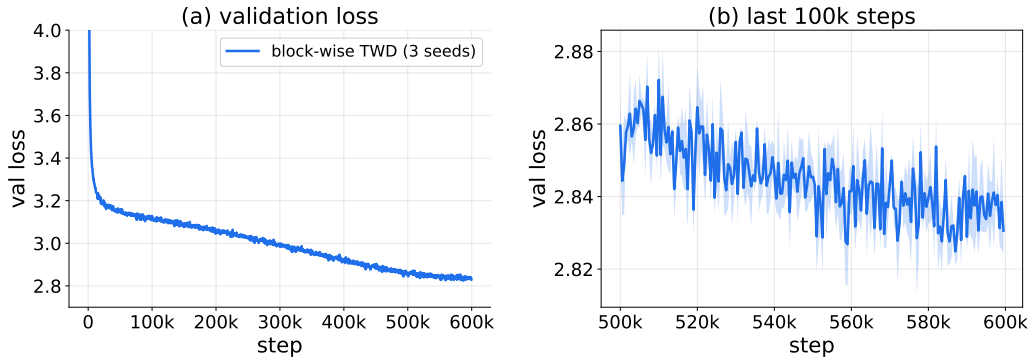

Figure B.5: **Reference: results from three seeds (1338, 1339, 1340) on a different compute environment for GPT-2 (124M).** The validation loss at 600k steps is 2.834 ($\pm$0.003) averaged over the three seeds. TWD runs stably across the three seeds.

For reference, Figure B.5 shows results from three seeds (1338, 1339, 1340) run on a compute environment different from that of Figure 4, Table B.9, and the throughput in Appendix D. The validation loss at 600k steps was 2.834 ($\pm$0.003) averaged over the three seeds, and TWD ran stably across the three seeds on GPT-2 (124M) as well; because the environment differs, these results are not merged with Figure 4 or Appendix D.

B.7 Criticality Analysis: Diagnostic Results in the Target-Energy Scale Sweep

B.7.1 Endpoint Diagnostics for Modular Multiplication and Fashion-MNIST

The target-scale dependence for modular multiplication is shown in Figure B.6. Overall, the same trend as for modular addition was obtained. In the low-energy regime, the representational change toward the rich side is large (c) and the control magnitude (d) and signal amplification (f) are also large, while the seed-to-seed variability and the post-onset accuracy drop increase (a,b), and at $s = 0.25$ only 1 of the 4 seeds reached grokking. In the intermediate regime all seeds reach quickly and stably (a,b), and in the high-energy regime the dynamics move toward the lazy side as CKA rises (c), the onset is delayed, and for $s \geq 15$ grokking was not reached within the observation interval (a). The main difference from addition is that the fastest onset appears at a slightly lower $s = 0.5$–0.75, while for $s = 0.5$ one seed showed a delayed onset. Thus the result that a fast and stable regime lies between excessive change toward the rich side and insufficient feature learning on the lazy side is common to addition and multiplication.

The target-scale dependence for Fashion-MNIST is shown in Figure B.7. As for CIFAR-10, the final test loss is smallest in the intermediate range $s \simeq 2$–2.5 and increases on both the low- and high-energy sides (b). On the other hand, the train accuracy and the generalization gap increase with s, and the change of the test accuracy is relatively small (a). In the representational change, the CKA of the hidden layer rises with s, whereas the CKA of the logits decreases slightly (c). Thus the scale dependence of the Jacobian diagnostics differs by architecture, and the best test loss cannot be explained by a single Jacobian condition. On Fashion-MNIST, a balance among the change of the hidden representation, the adaptation of the output representation, and overfitting appears in the intermediate region.

The relative control magnitude decreased monotonically for both the hidden layer and the output layer (d). In contrast, the energy of the uncontrolled biases showed an inverted-U shape, increasing until around $s \simeq 2$–3 and then decreasing (e), unlike CIFAR-10, where it increased on the high-energy side. Moreover, the largest and mean squared singular values of the Jacobian increased monotonically with s (f). This differs both from modular arithmetic, where they increase at the low- and high-energy ends, and from CIFAR-10, where they mostly decrease as s increases. Thus, on Fashion-MNIST, the best test loss cannot be explained by a single control magnitude, uncontrolled energy, or Jacobian condition, and a balance among the change of the hidden representation, the adaptation of the output representation, and overfitting appears in the intermediate region.

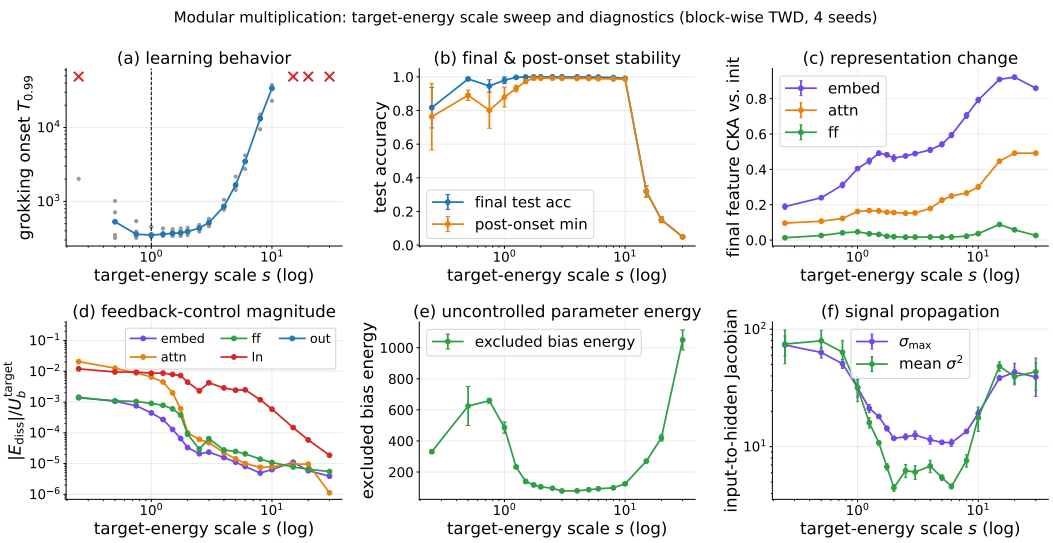

Figure B.6: **Target-scale sweep and diagnostics: modular multiplication** (block-wise TWD; mean ± standard deviation over four seeds). (a) Grokking onset $T_{0.99}$ (faint points = individual seeds, solid line = median over the 4/4-reached conditions, × = not reached, with the number reached out of four shown below); (b) final test accuracy and the post-onset minimum test accuracy (post-onset stability); (c) block-wise feature CKA (embedding, attention, feed-forward, and mean); (d) block-wise relative control magnitude $|E_{\text{diss}}|/U_b^{\text{target}}$; (e) energy of the excluded parameters (biases); (f) signal propagation (the $\sigma_{\max}$ and mean σ^2 of the input–hidden Jacobian). The panel layout is identical to Figure 5 (addition); (a) and (b) are task-specific performance metrics, and (c)–(f) are diagnostics common to all tasks.

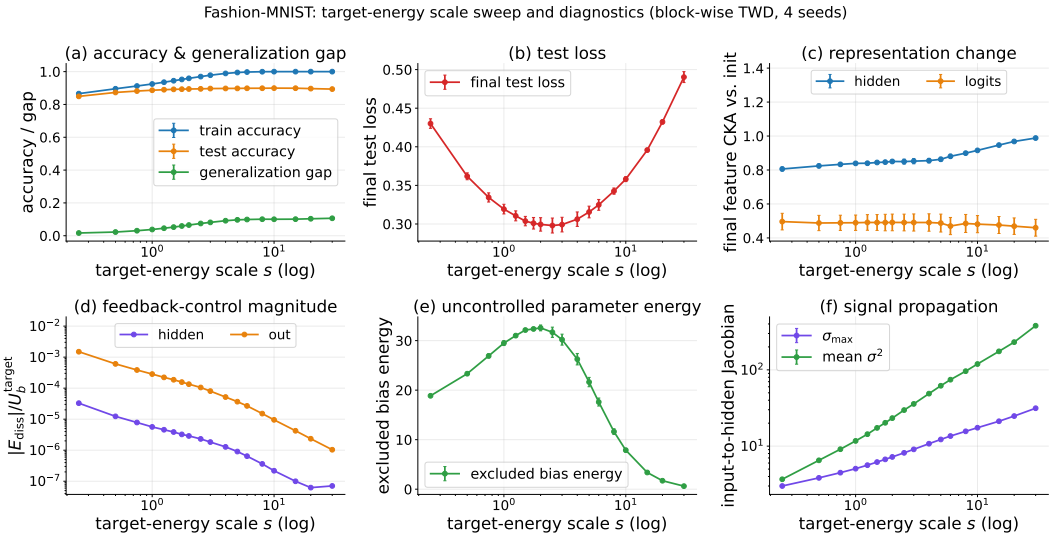

Figure B.7: **Target-scale sweep and diagnostics: Fashion-MNIST (Two-layer FFN)** (block-wise TWD; mean ± standard deviation over four seeds; only 3 seeds for s=6). (a) Train/test accuracy and generalization gap; (b) final test loss; (c) feature CKA relative to the initial representation (hidden and logits); (d) block-wise relative control magnitude $|E_{\text{diss}}|/U_b^{\text{target}}$ (hidden and out); (e) energy of the excluded parameters (biases); (f) signal propagation (the $\sigma_{\max}$ and mean σ^2).

B.7.2 Temporal Evolution of Signal Propagation

In addition to the scale diagnostics at the endpoint checkpoint, for modular addition we analyzed the signal-propagation properties during training, aligned to events. The target is all 18 scales from

$s = 0.25$ to 30, with four seeds each, for a total of 72 runs. Following the criteria of Appendix A.5.3, we analyzed a total of 319 checkpoints from epoch 0, the point at which train accuracy first reached 0.99, just before and just after the grokking onset, and the final state. Of the 72 runs, the point of reaching train accuracy 0.99 was obtained for 63 runs, and the checkpoints before and after the grokking onset for 56 runs.

Figure B.8 shows the largest singular value $\sigma_{\max}$ and the mean squared singular value $\langle \sigma^2 \rangle$ of the Jacobian at each learning event. At epoch 0, $\sigma_{\max}$ increased gradually from 1.59 at $s = 0.25$ to 4.55 at $s = 30$, and $\langle \sigma^2 \rangle$ from 0.709 to 1.11. By the time train accuracy reached 0.99, both metrics

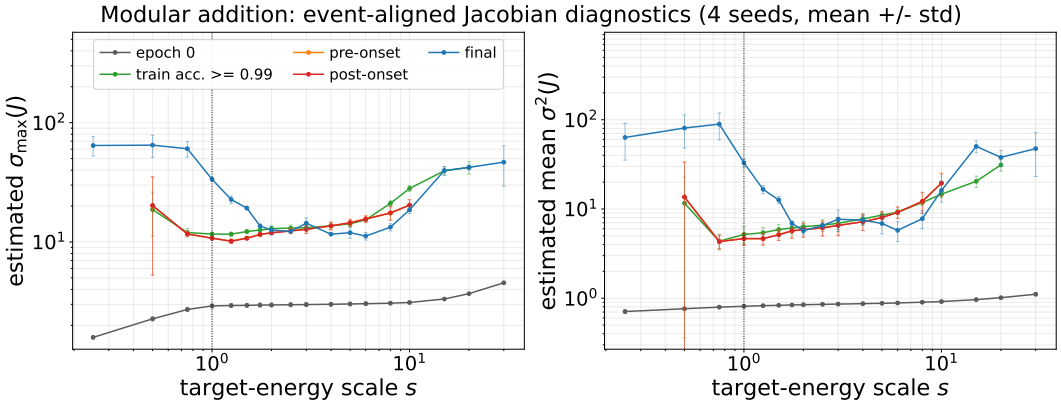

Figure B.8: **Temporal change of the Jacobian diagnostics for modular addition.** For all 18 scales and four seeds, we show (a) the estimated largest singular value $\sigma_{\max}(J)$ and (b) the estimated mean squared singular value $\langle \sigma^2(J) \rangle$ at epoch 0, at the time train accuracy reaches 0.99, just before and just after the grokking onset, and at the final checkpoint. Solid lines and points are seed averages, and error bars are the standard deviation across seeds. Runs that did not reach a given event were excluded from the aggregation, so some series are absent at s=0.25 and on the high-energy side. The vertical dotted line marks the reference scale s=1.

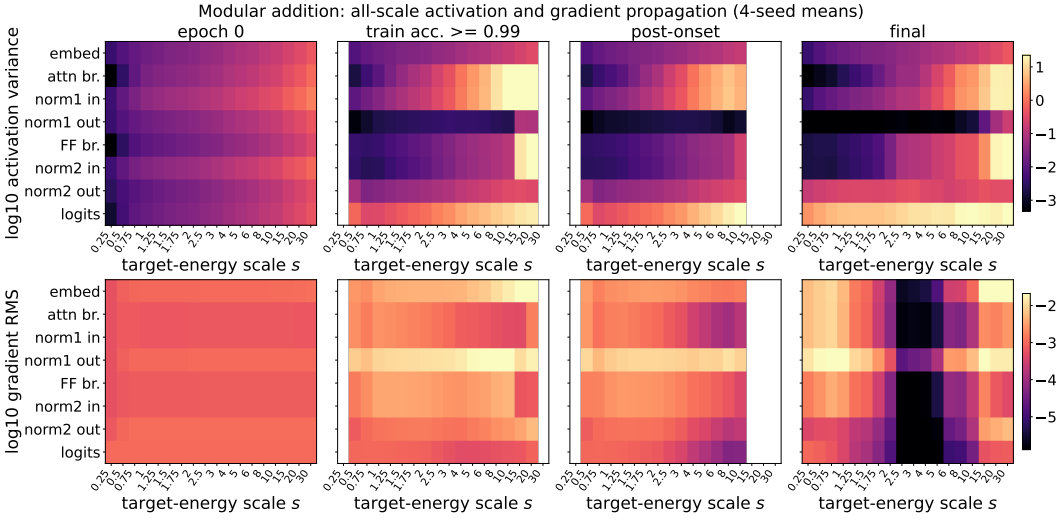

Figure B.9: **Temporal and scale dependence of the activation variance and gradient RMS for modular addition.** For all 18 scales and four seeds, we show the values at each computational stage at epoch 0, at the time train accuracy reaches 0.99, just after the grokking onset, and at the final state. The top row is the activation variance and the bottom row is the gradient RMS, with color the common logarithm of the four-seed average. Conditions that did not reach the grokking onset are shown blank. As with the Jacobian diagnostics, the signal-propagation properties depend on both the learning stage and the target energy, and in particular the differences across blocks widen after learning.

had increased for all reached runs. The rate of change from epoch 0 to the time of reaching train accuracy was, at the median, a factor of 4.54 for $\sigma_{\max}$ and 8.48 for $\langle \sigma^2 \rangle$. Thus the formation of the learned signal amplification is already progressing in the stage before the memorization of the training data is established.

The checkpoints just before and just after the grokking onset are in principle 10 epochs apart, but no discontinuous change was observed in the Jacobian diagnostics over this interval. The median ratio from just before to just after was 0.998 for both $\sigma_{\max}$ and $\langle \sigma^2 \rangle$. Thus, at least at the temporal resolution of this measurement, the grokking onset is not an event in which the global amplification factor of the Jacobian jumps instantaneously. Rather, test generalization emerges later on top of the signal-propagation state formed during the training-accuracy formation.

On the other hand, a clear scale dependence appeared in the level of the Jacobian. Under the onset-reached condition, $\sigma_{\max}$ was smallest around $s = 1.25$ and increased toward the high-energy side. Even on the low-energy side, the seed-to-seed variability was large at $s = 0.5$. In the final state this trend became even clearer, and both $\sigma_{\max}$ and $\langle \sigma^2 \rangle$ showed a U-shaped relationship: large at the low- and high-energy ends and small in the intermediate regime. For example, the average $\sigma_{\max}$ in the final state was 64.4 at $s = 0.25$, 11.2 at $s = 6$, and 46.7 at $s = 30$. This indicates that both a too-low target and a too-high target are accompanied by strong signal amplification, through different learning paths.

Figure B.9 shows the activation variance and the gradient RMS measured at the same checkpoints. At epoch 0 both vary relatively smoothly with the scale, but after train accuracy is reached the differences across computational stages widen. On the high-energy side, the activation variance of the attention branch, the feed-forward branch, and the logits becomes large. On the other hand, the gradient RMS in the final state does not change monotonically; it is small at intermediate scales and grows again at the low- and high-energy ends. This shows that the target energy does not merely change the overall gain of the whole network uniformly but changes the allocation of activations and gradients across blocks.

From this temporal analysis, we find that the learning regime prescribed by the target energy is not formed only at the moment of the grokking onset but is gradually formed from the process leading to the memorization of the training data. Moreover, although the intermediate regime in which good grokking is observed is not a strict dynamical-isometry regime satisfying $\sigma_{\max} \approx 1$, it avoids the extreme signal amplification seen at the low- and high-energy ends. This result therefore supports interpreting the neighborhood of the reference energy not as a single Jacobian condition but as a finite region in which representational change, signal propagation, and learning stability are balanced.

B.7.3 Singular-Value Distribution

The largest singular value and the mean squared singular value alone cannot distinguish a concentration onto a few large singular values from a uniform amplification of the whole spectrum. We therefore explicitly constructed the 256×256 Jacobian for a fixed single diagnostic sample, for all 319 checkpoints used in the temporal diagnostics, and obtained all singular values by a full SVD. Figure B.10 shows the quantile bands of the singular-value distribution at each learning event, the normalized effective rank, and the cumulative distribution in the final state.

From epoch 0 to the time train accuracy reached 0.99, the spectrum did not amplify uniformly but became strongly anisotropic. For the 63 runs for which both time points were obtained, the largest singular value became a median factor of 4.51, while the 10th percentile of the spectrum became a factor of 0.192 and the normalized effective rank a factor of 0.387. Thus, in the process leading to the memorization of the training data, amplification of a few directions and contraction of many directions proceed simultaneously.

On the other hand, no discontinuous change was seen in the full spectrum just before and just after the grokking onset either. For the 56 runs, the median ratio from just before to just after was 0.991 for the largest singular value, 0.988 for the spectrum median, 1.000 for the 90th percentile, and 0.997 for the effective rank. There was also no substantial change in the fraction of near-zero singular values. This result shows that the grokking onset does not arise from an instantaneous reorganization of the Jacobian spectrum, but on top of the anisotropic signal-propagation state formed beforehand.

From after the onset to the final state, many singular values moved toward zero while the upper end of the spectrum was largely maintained. The median rate of change from after the onset to the final state was a factor of 1.10 for the largest singular value and 1.10 for the 90th percentile, whereas the 10th percentile was 0.0050 and the spectrum median 0.0217. In addition, the fraction of singular values with $\sigma < 0.1$ increased by a median of 0.455. Thus, in the late stage of learning, the effective dimension of the Jacobian shrinks while a few high-gain directions remain.

The distribution in the final state depended strongly on the target energy. At the low-energy end $s = 0.25$, 77.1% of the singular values were below 0.1, while the largest singular value was 76.4 and the normalized effective rank was 0.024. This represents a state in which most of the spectrum contracts while only a few directions are strongly amplified. In the intermediate regime, the largest singular value was suppressed and the effective rank also relatively recovered; for example, at $s = 4$ the largest singular value was 11.9 and the normalized effective rank was 0.133. At the high-energy end $s = 30$, the spectrum median increased to 0.985, while 39.2% of the singular values exceeded 2 and the largest singular value was 32.1. Thus the high-energy condition retains many propagation directions but at the same time includes strongly amplified directions.

From the above, the region of good learning cannot be characterized as dynamical isometry, in which $\sigma_i \approx 1$ holds in all directions. On the low-energy side, a reduction in rank and a concentration of amplification onto a few directions appear, and on the high-energy side, a broad propagation dimension and many amplified directions appear. The intermediate regime avoids these two extreme spectral structures, but neither does a single singular-value statistic alone determine the grokking speed. This result supports interpreting the critical region in this study not as a strict isometry point but as a finite region in which signal amplification, effective dimension, representational change, and learning stability are balanced.

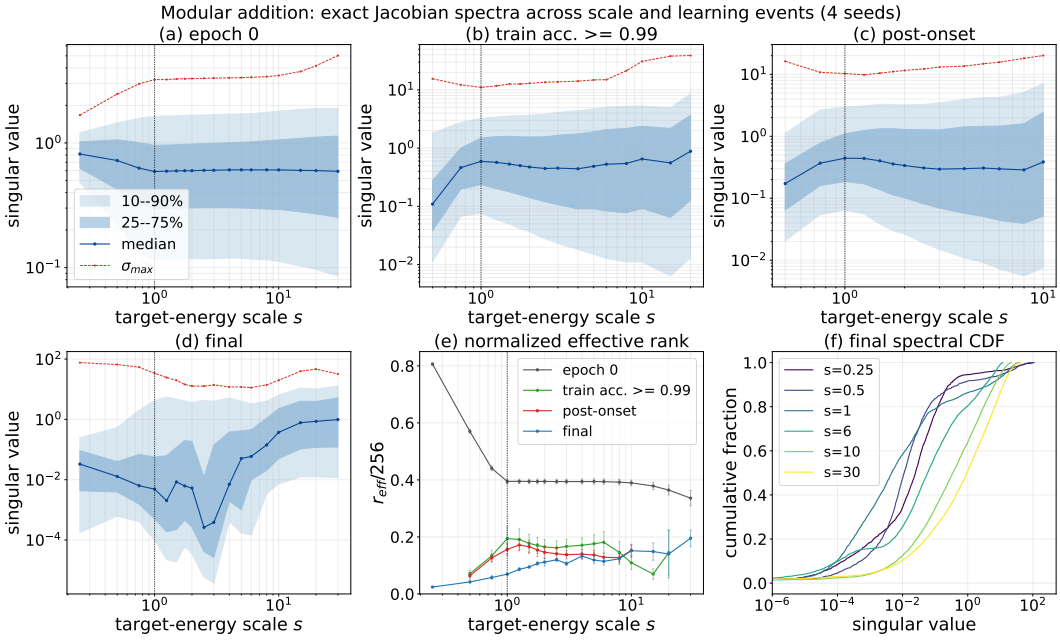

Figure B.10: **Temporal and scale dependence of the exact Jacobian spectrum for modular addition.** For all 18 scales and four seeds, we performed a full SVD of the 256×256 Jacobian for a fixed single diagnostic sample. Singular-value distributions at (a) epoch 0, (b) the time train accuracy reaches 0.99, (c) just after the grokking onset, and (d) the final state. The dark band is the 25–75% quantile, the light band the 10–90% quantile, the blue line the median, and the red dashed line the largest singular value. (e) The normalized effective rank $r_{\mathrm{eff}}/256$ computed from the squared contributions of all singular values. (f) The cumulative distribution of the final spectrum at representative target scales; each curve aggregates the singular values of the four seeds. Conditions that did not reach the corresponding learning event are omitted from (a)–(e).

Table C.1: **Comparison with existing weight-decay methods.** TWD infers the weight-decay coefficient before the update, accounting for the actual base-optimizer update, so as to track a block-wise target energy defined from the model structure.

Method	Observable / target	Control mechanism	Granularity	Actual optimizer update	Reference
AdaDecay	Gradient statistics; no target	Adapts decay strength	Parameter-wise	Not modeled	(Nakamura & Hong, 2019)
OUIDecay	Activation statistics; no target	Redistributes decay across layers	Layer-wise	Not modeled	(Fernández-Hernández et al., 2026)
LARS/LAMB	$\|w\|/\|g\|$; no target	Scales the learning rate	Layer-wise	Step size only	(You et al., 2017; 2020)
max-norm / projection	Fixed norm upper bound	Projects after the update	Unit/matrix-wise	Applied post-update	(Srivastava et al., 2014)
CPR	Norm upper bound	Adapts the constraint multiplier	Matrix/group-wise	No inversion	(Franke et al., 2024)
AdamWN	Externally specified target radius	Draws the weight norm toward the target	Global	Partial	(Loshchilov, 2023)
OrthoGrad	Weight and gradient directions	Orthogonally projects the gradient	Parameter-wise	Reshapes the gradient	(Prieto et al., 2025)
TWD (ours)	$U_b^{\mathrm{target}}=s_b U_b^{\mathrm{ref}}$	Pre-update quadratic inversion	Global / block-wise	Actual AdamW/SGD update	—

C Comparisons with Existing Methods and Validation of the Proposed Method

In this appendix, we compare the proposed method with existing techniques and validate it. First, Table C.1 gives an algorithmic comparison of existing techniques and the proposed method. We then validate the effectiveness of the proposed method through comparisons with AdamWN, OrthoGrad, and post-projection.

C.1 AdamWN: Comparison with Target-Norm Control

Figure C.1 shows the comparison of AdamWN and block-wise TWD. We fixed the AdamWN target-norm ratio to $r_t = 0.3$ and swept $k_t \in \{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$ with four seeds for each

Table C.2: **Selection of the tuned AdamWN baseline (k_t) for modular addition and multiplication (4 seeds, 40k epochs).** With r_t=0.3 fixed, we swept k_t for each task. Among the candidates for which all seeds grok while maintaining the final test accuracy, we selected the k_t with the smallest onset for each task. The adopted value for each task is shown in bold. We adopted k_t=10^{-3} for addition and k_t=10^{-1} for multiplication. Onset and final acc are averages over four seeds. NR indicates that only some of the four seeds reached grokking (including collapse of the final accuracy).

k_t	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}
Addition onset	NR	NR	**3,062**	12,165	23,010
Addition final acc	0.752	0.752	**1.000**	**1.000**	**1.000**
Multiplication onset	**458**	605	2,650	10,730	19,075
Multiplication final acc	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**

Softmax collapse occurs for $k_t \leq 10^{-4}$ (the maximum collapse is 0.18 at 10^{-4} and 0.48 at 10^{-5} for addition, and 0.15 at 10^{-4} and 0.65 at 10^{-5} for multiplication) and does not occur for $k_t \geq 10^{-3}$. For addition, at $k_t \geq 10^{-2}$ one of the four seeds does not grok and the final accuracy collapses, so the onset is marked NR.

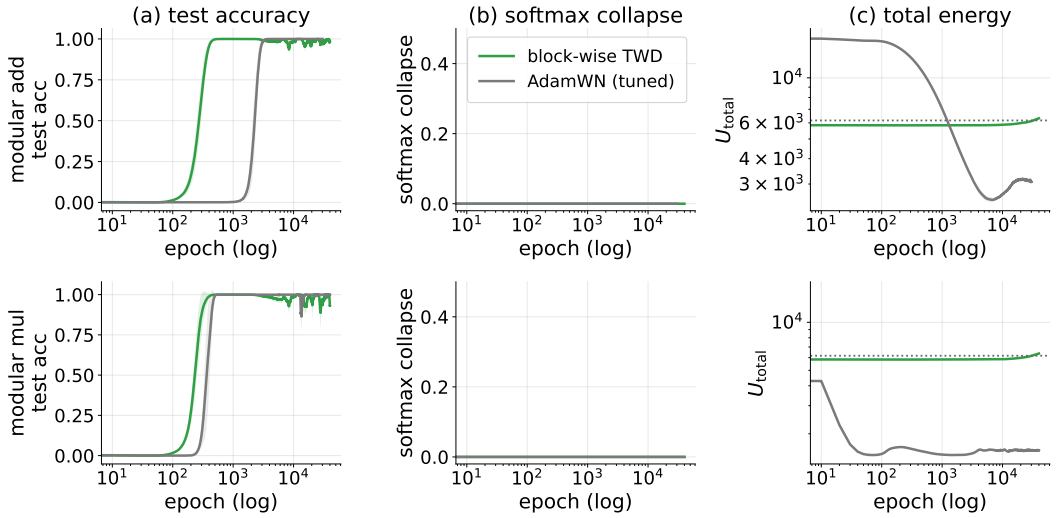

Figure C.1: **Comparison with AdamWN on modular addition and multiplication** (mean ± standard deviation over four seeds; 40k epochs). AdamWN is fixed at r_t=0.3, and for each task we adopt the fastest setting in which all seeds reach grokking, selected from a 5-point k_t sweep (k_t=10^{-3} for addition, k_t=10^{-1} for multiplication). Tuned AdamWN avoided softmax collapse and grokked for all seeds, but its onset was slower than block-wise TWD by a factor of approximately 6.8 for addition and approximately 1.24 for multiplication. (c) shows the total internal energy U_{total}; the dashed line marks the reference energy U^{ref} (U/U^{target}=1).

task. Among the settings where all four seeds reached grokking while maintaining the final accuracy, we selected the value with the smallest average onset. As a result, we adopted $k_t = 10^{-3}$ for addition and $k_t = 10^{-1}$ for multiplication (Table C.2). This is a condition tuned favorably to AdamWN using the evaluation results. Tuned AdamWN avoided softmax collapse on both tasks and grokked at 3,062 epochs for addition and 458 epochs for multiplication. The onset of block-wise TWD was 448 and 370 epochs, respectively; thus block-wise TWD was faster by a factor of approximately 6.8 for addition, while for multiplication the difference narrowed to a factor of approximately 1.24. Therefore, even global norm control can achieve a speed close to TWD on multiplication if an appropriate control rate is chosen, but a clear advantage of functional block-wise control remained for addition.

C.2 OrthoGrad: Comparison with Gradient Orthogonalization

OrthoGrad completely removes the gradient component parallel to the weights and renormalizes the orthogonalized gradient to the original gradient norm. In this experiment we followed the public implementation of Prieto et al. (2025) and did not introduce any tunable hyperparameters specific to OrthoGrad. We used the same AdamW base optimizer as TWD, and the other training conditions, including the learning rate, the moment coefficients, and the logit scaling, were also shared.

The comparison for modular addition and multiplication is shown in Figure C.2 (the vertical dashed line marks 40k epochs, the endpoint of the ordinary experiment). Up to 40k epochs, OrthoGrad and block-wise TWD grokked at roughly the same speed and no collapse occurred, but the total energy of OrthoGrad had already reached approximately 24–25 times the reference at this point. When training was extended to 100k epochs, the total energy of OrthoGrad grew to approximately 122 times the reference for addition and approximately 127 times for multiplication, and softmax collapse increased sharply at a mean of about 86k and about 79k epochs, respectively. In contrast, block-wise TWD kept the total energy near the target energy U^{target} and no collapse occurred. The grokking speeds of the two methods are comparable, but the long-time energy state and the stability of the output distribution differ greatly.

The comparison for Fashion-MNIST is shown in Figure C.3. While block-wise TWD keeps the total energy near the target energy U^{target}, the total energy of OrthoGrad grew to approximately 3.1 times the reference. At 500 epochs, both methods reached a comparable test accuracy of about 0.89,

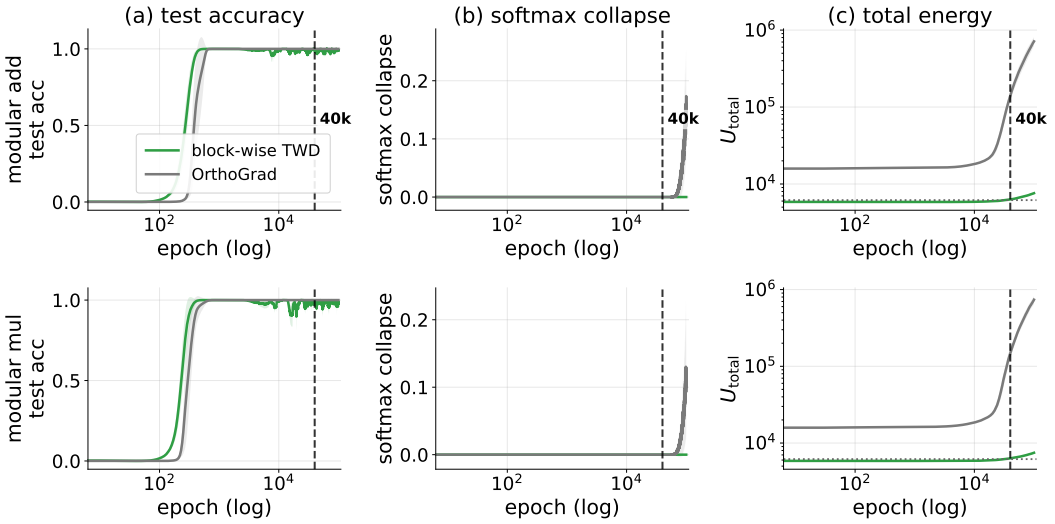

Figure C.2: **Comparison with OrthoGrad on modular addition and multiplication** (mean $\pm$ standard deviation over four seeds; 100k epochs). The top row is modular addition and the bottom row is multiplication; each column shows the test accuracy, softmax collapse, and total energy U_{total}. The vertical dashed line marks 40k epochs, the ordinary training budget. The grokking onsets of the two methods are comparable, but for OrthoGrad the total energy grows to approximately 122–127 times the reference and softmax collapse recurs in the later stage of training, whereas block-wise TWD stays near the target energy and no collapse occurs.

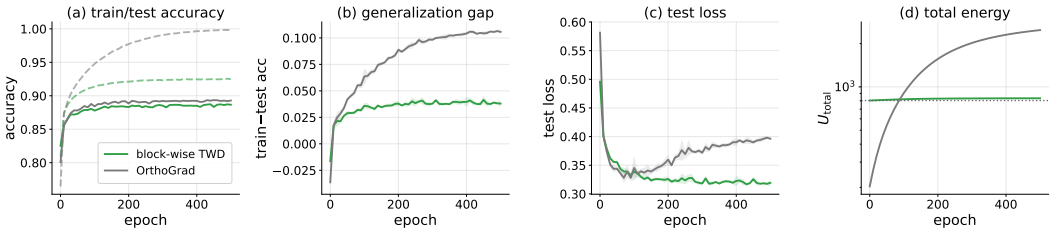

Figure C.3: **Comparison with OrthoGrad on Fashion-MNIST (Two-layer FFN)** (mean $\pm$ standard deviation over four seeds; 500 epochs). Each panel shows (a) train/test accuracy, (b) the generalization gap, (c) the test loss, and (d) the total energy U_{total}. While block-wise TWD suppresses the increase of the test loss, OrthoGrad's test loss increases together with the growth of the total energy.

but block-wise TWD had a generalization gap of 0.038 and a test loss of 0.319, whereas OrthoGrad remained at a gap of 0.106 and a test loss of 0.396. While block-wise TWD suppresses the increase of the test loss, OrthoGrad's test loss increases together with the growth of the total energy.

C.3 Post-projection: Comparison with Post-update Projection

For TWD, we infer a block-wise weight-decay coefficient before each update and then perform the AdamW update using that coefficient. For post-projection, we first perform an AdamW update under the same conditions and then apply PBWN-based unit-norm projection with strength 1.0 (Huang et al., 2017). This projection does not target the architecture-aware energy U^{target} used by TWD. The other training conditions were shared.

Figure C.4 compares modular addition and multiplication. Post-projection reached grokking onset faster than block-wise TWD (328 versus 448 epochs for addition and 278 versus 370 epochs for multiplication), but softmax collapse increased sharply in the later stages of training. Block-wise TWD did not exhibit collapse. Moreover, the total energy under post-projection remained at approximately half of TWD's target U^{target}, whereas block-wise TWD maintained the total energy near U^{target}.

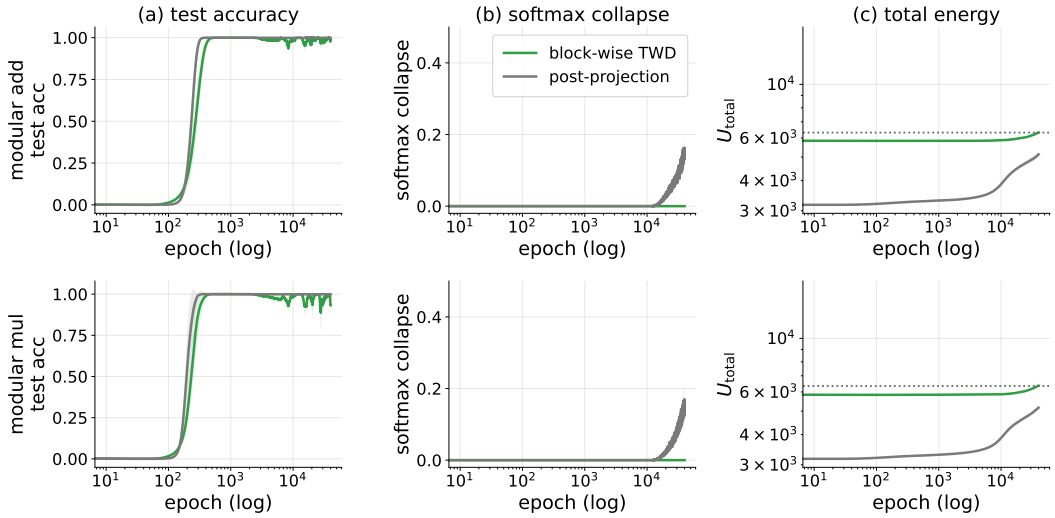

Figure C.4: **Post-projection (`unit_norm`) versus block-wise TWD on modular addition and multiplication.** The top row shows addition and the bottom row multiplication. (a) Test accuracy; (b) softmax collapse; (c) total energy U_{total}. Both the green and gray curves are observed U_{total}; only the dotted line marks TWD's target energy U^{target}.

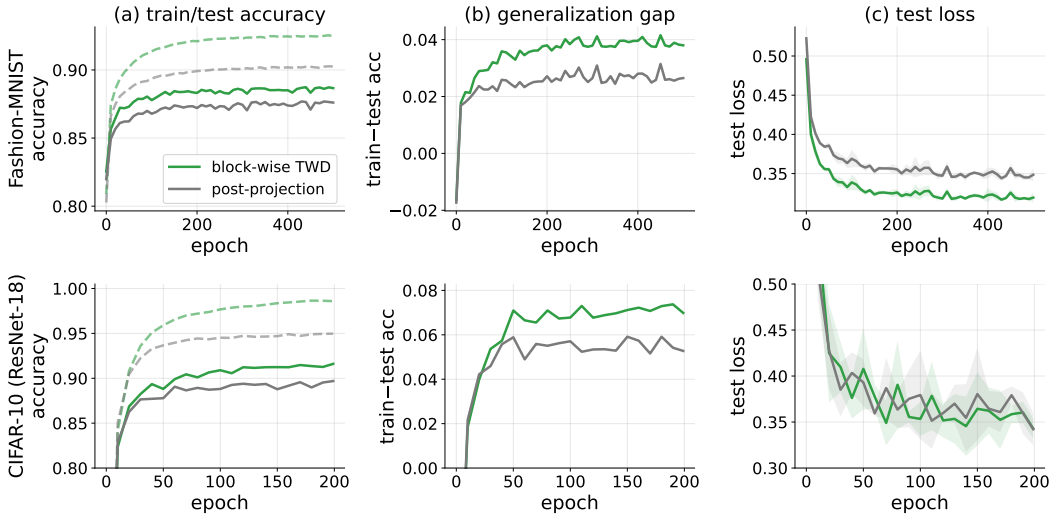

Figure C.5: **Post-projection (`unit_norm`) versus block-wise TWD on Fashion-MNIST and CIFAR-10.** The top row shows Fashion-MNIST and the bottom row CIFAR-10. (a) Train/test accuracy (solid = test, dashed = train); (b) generalization gap; (c) test loss.

Figure C.5 compares Fashion-MNIST and CIFAR-10. Post-projection had a smaller generalization gap, but block-wise TWD achieved higher final test accuracy on both tasks (0.887 versus 0.876 on Fashion-MNIST and 0.916 versus 0.897 on CIFAR-10). The CIFAR-10 test losses were nearly equal (approximately 0.34). Thus, although post-projection accelerates grokking, it exhibits softmax collapse in the later stages and falls short of block-wise TWD in final test accuracy on the vision tasks. Because the two methods also use different control targets, this comparison does not isolate the effect of update order alone.

We also test a projection variant, post-projection (U^{target}), that after each base update isotropically rescales the squared Frobenius norm of each tensor to the same U^{target} that block-wise TWD uses—the same block-wise, architecture-aware target, with the same architecture-based choice of which parameters are controlled (for example, biases are excluded from control). The results are shown

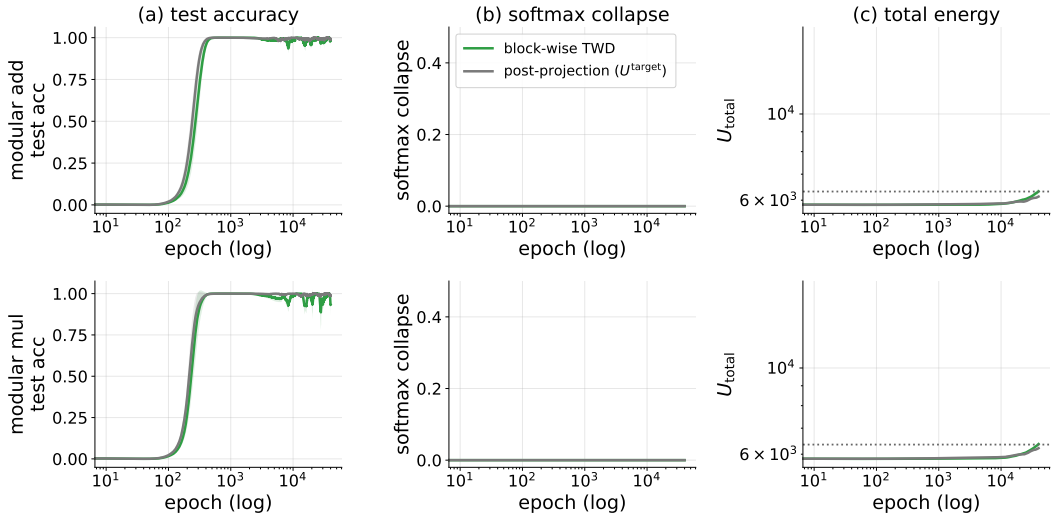

Figure C.6: **Post-projection (U^{target}) versus block-wise TWD on modular addition and multiplication.** The top row shows addition and the bottom row multiplication. (a) Test accuracy; (b) softmax collapse; (c) total energy U_{total}.

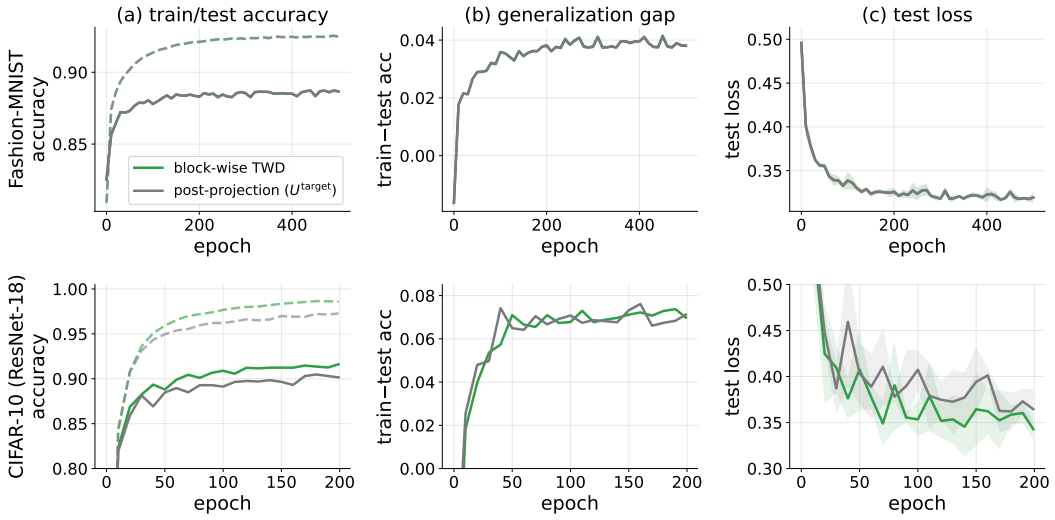

Figure C.7: **Post-projection (U^{target}) versus block-wise TWD on Fashion-MNIST and CIFAR-10.** The top row shows Fashion-MNIST and the bottom row CIFAR-10. (a) Train/test accuracy (solid = test, dashed = train); (b) generalization gap; (c) test loss.

in Figures C.6 and C.7. Strikingly, post-projection (U^{target}) and block-wise TWD give essentially identical results on the modular-arithmetic tasks (one-layer Transformer) and on Fashion-MNIST (Two-layer FFN): the grokking onset, the softmax collapse (no collapse in either case), and the total energy (kept near U^{target}) all overlap, and on Fashion-MNIST the final test accuracy and test loss are identical (0.887 / 0.319). Only on CIFAR-10 (ResNet-18) do the two differ, and there block-wise TWD is better (test accuracy 0.916 vs. 0.901; test loss 0.343 vs. 0.365). Whereas post-projection enforces the target as a one-shot isotropic rescaling after each update, block-wise TWD controls weight decay sequentially, block by block, inside the update. On relatively simple architectures such as the one-layer Transformer and the Two-layer FFN, the two approaches yield equivalent results. As for CIFAR-10 (ResNet-18), where a difference appears, Fashion-MNIST is also trained with SGD yet shows no such difference, which suggests that the cause is not the optimizer but the architectural complexity. In ResNet-18, which has residual connections and Batch Normalization,

the energy balance across blocks affects the learning trajectory, so block-wise TWD, which applies sequential per-block control inside the update, proves more effective.

D Computational Cost and Execution Environment

In this appendix, we evaluate the computational cost of TWD on the grokking tasks and on the GPT-2 (124M) OpenWebText pretraining. Note that this implementation does not include any special acceleration or optimization for parallel training, and reducing the computational cost and increasing the throughput are left for future work.

D.1 Measurement Environment and Procedure

The grokking tasks were run on one AMD Instinct MI300X and GPT-2 on four, executed on ROCm. For GPT-2 we used bfloat16, DistributedDataParallel, and RCCL.

For the grokking tasks, we decomposed the wall-clock time of 40k epochs into forward/backward, optimizer, evaluation, and so on, and averaged over four seeds. For GPT-2, we recorded training in several execution intervals and converted the number of steps advanced in each interval to steps/hour, averaging across intervals (about 44k–49k steps per interval). We saved the model, optimizer, scheduler, and TWD control state to a checkpoint and resumed from it. For the grokking tasks, to measure the pure processing time, we kept only the main procedure (Figure A.1) and excluded the time to compute the internal metrics for logging.

D.2 Computational Cost on Grokking Tasks

The execution time of 40k epochs is shown in Figure D.1. AdamW took 163.3 seconds, global TWD 251.6 seconds, and block-wise TWD 278.3 seconds, with relative times of 1.00, 1.54, and 1.70, respectively. The forward/backward and evaluation times are nearly constant, whereas the TWD overhead concentrates in the optimizer step. The TWD portion above the optimizer is the time for the sequential inversion of the weight-decay coefficient, corresponding to about $+88$ seconds for global TWD and about $+109$ seconds for block-wise TWD.

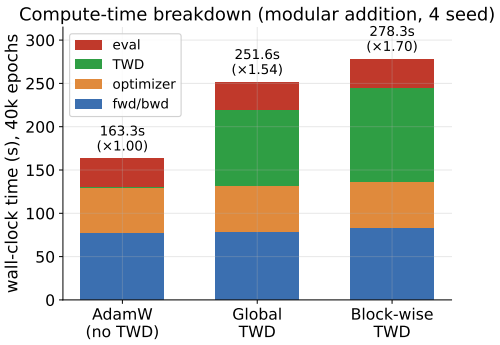

Figure D.1: **Breakdown of the computation time on the grokking tasks (modular addition, 40k epochs, average over four seeds).** The numbers on the bars are the total time and the ratio to AdamW.

D.3 Throughput in GPT-2 (124M) Pretraining on OpenWebText

The number of steps processed per hour (throughput) during the 600k-step run is shown in Figure D.2. The original nanoGPT achieved 8,100 steps/hour ($n = 12$ intervals, standard deviation 98), and block-wise TWD (architecture-aware reference, LayerNorm excluded, $s = 1$) achieved 7,683 steps/hour ($n = 13$ intervals, standard deviation 95), a relative time of about 1.05. Converting this to the processing time of 600k steps, nanoGPT corresponds to about 74.1 hours and block-wise TWD to about 78.1 hours. All conditions share the same model, data, effective batch size, scheduler, precision, and GPU configuration.

Whereas the computational cost on the grokking tasks was 1.70 times that of AdamW, on GPT-2 it was about 1.05 times that of nanoGPT, and the additional computation became relatively small compared with the forward/backward pass. However, a general scaling law cannot be derived from the results at two model scales. For the two settings measured in this study, the relative overhead was smaller on the GPT-2 side.

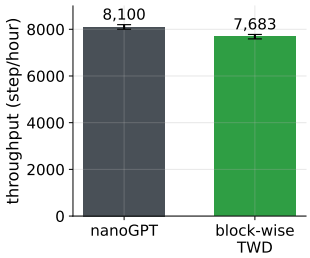

Figure D.2: **Training throughput on GPT-2 (124M).** Training logs in the same environment (four AMD Instinct MI300X, bfloat16) are converted to steps/hour for each execution interval (about 44k–49k steps advanced per interval) and averaged across intervals (larger is faster). The throughput of block-wise TWD is $7{,}683$ steps/hour (n=13 intervals, standard deviation 95), only about a 5% reduction relative to nanoGPT's $8{,}100$ steps/hour (n=12, standard deviation 98). Error bars are the standard deviation across intervals.